Universitetet i Oslo
Institutt for informatikk

Internett og ekspertsystemer: Som fugl Føniks opp av asken?

Steffen Heim

Masteroppgave

1. November 2007

Innhold

1 Innledning og problemstillinger

Moderne samfunn er i større grad enn noen gang tidligere avhengige av eksperter. Jo mer komplisert et samfunn blir, jo mer viten man samler, jo mer trengs mennesker som spesialiserer seg på et lite aspekt av et større fagfelt, som så å si kan mye om stadig mindre. Denne utviklingen er naturlig fordi det finnes så mye viten innenfor et fagfelt at det ikke er mulig for et enkelt menneske å ha oversikten over alt. Altså trenges det flere som kan hver sin lille bit (Ericsson & Charness 1994).

Eksperter har gjerne en utdannelse innenfor sitt fagfelt som utgangspunkt. Denne utdannelsen alene er imidlertid sjelden nok til å bli anerkjent som ekspert. Med ekspertise forbinder man som regel også lang erfaring i å bruke ens fagkunnskap til å løse problemer. Utdannelse og erfaring gjør eksperten i stand til å løse problemer som lekfolk ikke makter. For eksempel er en lege i stand til å stille en nøyaktigere diagnose, gitt en liste av symptomer, enn en som ikke har beskjeftiget seg med medisin. Presumptivt skulle en lege med ti års erfaring også kunne stille en mer nøyaktig diagnose enn en nyutdannet lege.

Eksperters evne til å løse problemer basert på sin kunnskap og erfaring gjør dem ettertraktet. Eksperter konsulteres stadig av andre som møter problemer de selv ikke er i stand til å løse. Svært ofte er det også pålagt å bruke eksperter. For eksempel må det, før man skal bygge en tunnel, foreligge en geologisk vurdering av fjellet tunnelen skal gå gjennom.

Avhengigheten av eksperter har naturligvis noen ulemper. Det tar lang tid før noen blir ekspert på et fagfelt. Å ha beskjeftiget seg intensivt med et domene i ti år blir ofte ansett som et minimum for å kunne prestere på ekspertnivå (Ericsson & Charness 1994). For virkelig prominente eksperter er det kanskje snakk om enda mer. Dette gjør at det finnes forholdsvis få eksperter i de ulike spesialiserte grenene av et fagområde. Så jo mer spesialisert og nisjeaktig en slik subdisiplin er, jo færre eksperter er det mulig å oppdrive. Dette leder igjen til at det kan finnes for få eksperter i forhold til behovet for akkurat denne typen ekspertise. Dette fører naturligvis til at ekspertene kan ta seg mer betalt for sine tjenester og slik kan det bli en dyr affære å innhente ekspertråd.

Å kunne etterligne eksperter ved hjelp av dataprogrammer har derfor vært et stort tema innenfor forskningen rundt kunstig intelligens. Man tenkte seg programmer som etter hvert ville kunne erstatte de menneskelige ekspertene og slik gjøre dyr og sjelden ekspertkunnskap lettere tilgjengelig for allmennheten.

Disse programmene, som man med stor entusiasme begynte å lage på 1960-tallet, ble kalt for ekspertsystemer eller kunnskapsbaserte systemer. Disse betegnelsene gjenspeiler tanken om at systemer av denne typen etterligner eksperter ved hjelp av ekspertens kunnskap. Typisk er disse systemene laget med en eller flere eksperter som forbilde. Uansett hvilket fagområde man vil lage et ekspertsystem for forsøker man å få tak i ekspertens kunnskap og omformulere denne kunnskapen til regler som ekspertsystemet kan bruke.

Etter hvert som utviklingen av ulike ekspertsystemer gikk framover viste det seg at alle ekspertsystemer hadde en hel del egenskaper til felles. For eksempel må et ekspertsystem ha en måte å komme frem til konklusjoner på, en slutningsmekanisme av noe slag. Dette gjelder enten kunnskapen systemet jobber med handler om indremedisin eller geologi. Man begynte derfor å samle disse felles komponentene i egne programmer, såkalte ekspertsystemskall. Et ekspertsystemskall er i praksis et program som inneholder alle tjenestene som

trengs for å kunne trekke konklusjoner ut fra en mengde kunnskapsregler. Det som ikke finnes i et slikt skall er nettopp kunnskapen. Slik skal et ekspertsystemskall i teorien kunne bruke kunnskap fra ulike fagfelt gitt at denne kunnskapen er på et format som programmet forstår. Slik blir ekspertsystemskallet et mer generelt problemløsningsprogram som er avhengig av spesifikk fagkunnskap for å kunne løse problemer.

Ekspertsystemtradisjonens store visjoner om å lage kunstige eksperter viste seg imidlertid vanskelige å gjennomføre i praksis. Ekspertsystemene slet med betydelige problemer av ulike slag. Hvordan skulle man for eksempel holde systemets regler oppdatert med ny kunnskap etter at brukeren hadde fått programmet? Hvordan gjøre systemene fysisk lett tilgjengelige når systemet nødvendigvis måtte være installert på en datamaskin som igjen nødvendigvis var bundet til et bestemt sted? Det viste seg tungvint å få tak i ekspertenes kunnskap. Hvordan kunne man gjøre dette mer effektivt? Hvordan skulle systemer reagere dersom de ble bedt om å løse problemer som gikk utover deres begrensede kunnskap? Hvordan skulle systemet forholde seg dersom brukeren var usikker på den informasjonen som systemet spurte etter eller dersom brukeren manglet informasjon? Dette siste problemet er som vi skal se spesielt relevant for ekspertsystemer som skal løse medisinske problemer.

I tillegg til problemer med ekspertsystemene selv ble også selve ekspertsystemkonseptet utsatt for kritikk fra bedømmings- og beslutningspsykologien, en underdisiplin av psykologi. Her mente man at menneskelige eksperter slett ikke er så gode som man ofte vil ha det til og at man ofte kan overgå eksperters prestasjoner med svært enkle statistiske metoder. Så hvorfor da bruke tid og krefter på å lage programmer som bare etterligner ekspertene når man ved enkle midler kan oppnå resultater som overgår dem?

Problemer som disse gjorde at interessen for ekspertsystemene avtok på 1980-tallet.

1.1 Problemstilling

På 1990-tallet kom imidlertid Internett. Dette verdensomspennende nettverket øker stadig i omfang og i løpet av de siste årene har det dukket opp en rekke internettbaserte ekspertsystemer. Det virker derfor som om Internett gir ekspertsystemteknologien nye muligheter. Oppgavens hovedproblemstilling er derfor: I hvilken grad kan Internett bidra til å løse problemene ekspertsystemene hadde på 1980-tallet?

I tillegg til å drøfte denne problemstillingen, er det et mål med denne oppgaven å lage et enkelt ekspertsystemskall. Dette gjøres både som en programmeringsøvelse og for å få en følelse med hvordan et slikt skall kan fungere.

1.2 Oppgavens struktur

Først, i kapittel to, skal jeg forklare hva ekspertsystemer er og hvordan de fungerer. I kapittel tre gir jeg en oversikt over problemene med ekspertsystemer på 1980-tallet, før Internett. I kapittel fire drøfter jeg oppgavens hovedproblemstilling, altså i hvilken grad Internett kan bidra til å løse eller redusere problemene ekspertsystemfeltet hadde på 1980-tallet. Jeg sammenfatter og konkluderer i kapittel fem.

Det lille ekspertsystemskallet jeg har utviklet i beskrives i appendiks A. En liten bruksanvisning for hvordan man får startet opp systemet finnes i appendiks B, mens kildekoden er gjengitt i appendiks C.

2 Om ekspertsystemer

På midten av 1950-tallet oppsto retningen kunstig intelligens innenfor informatikk. Dette var en fagretning der man var opptatt av om og hvordan datamaskiner kunne programmeres til å oppvise menneskelig, eller intelligent, atferd.

Denne interessen for å lage intelligent programvare var dels motivert ut i fra tanken om å kunne simulere menneskelige tankeprosesser. Ved hjelp av slike simuleringer håpet man å kunne danne mer eksakte teorier om hvordan mennesker faktisk tenker.

Dels var denne interessen motivert ut fra ønsket om å lage intelligente systemer, uavhengig om disse var simuleringer av menneskelig atferd eller ikke. Uansett motivasjon kom en del av kunstig intelligens forskningen til å dreie seg om problemløsning (Buchanan, Davis & Feigenbaum 2006).

De første problemløsningsprogrammene som kunstig intelligenstradisjonen forsøkte å frembringe var generelle problemløsere. Dette var programmer som i prinsippet skulle kunne løse et hvilket som helst problem så lenge dette problemet var gitt på et format som programmet forsto. Interessen for slike generelle programmer kom av tanken om at en generell problemløsningsevne er en viktig komponent i intelligent atferd. Premisset man opererte ut fra her var at noen er bedre til å løse problemer fordi de har en bedre generell problemløsningsevne enn andre.

Et kjent eksempel på et slikt generelt problemløsningsprogram er Herbert Simons General Problem Solver, eller GPS (Norvig 1992, Buchanan et al. 2006). Dette programmet tok mål av seg å løse problemer uavhengig av fagområde eller tema. Brukeren av GPS kunne definere et mål som programmet skulle forsøke å oppnå. Videre måtte brukeren opprette ulike objekter og operasjoner som kunne gjennomføres på disse objektene. For eksempel kan man definere et bilobjekt og et togobjekt. En operator kan være den handlingen det er å ta toget eller reparere bilen. Objektene og operatorene representerer slik det universet GPS «tenker» i. Når alle relevante elementer i dette universet var på plass, nermet GPS seg målet ved å skrittvis utføre de operasjonene som førte nærmere den ønskede måltilstanden.

Sett at det aktuelle problemet er å komme seg til universitetet. Sett videre at det kun finnes to måter å komme seg til universitetet på. Man kan enten ta tog eller kjøre sin egen bil. Det viser seg imidlertid at toget ikke går. GPS vil da gå videre til bilobjektet. Det viser seg at bilen ikke starter. Da vil GPS, gitt at de tilsvarende objektene og operatorene er definert, først forsøke å løse underproblemet med den defekte bilen, før den fortsetter med hovedproblemet.

På denne måten klarte GPS å løse enkle problemer innenfor områder som spill og matematikk. Satt på prøve av litt større og kompliserte problemer, der kombinasjonsmulighetene var større og hvor det er vanskelig eller umulig å formalisere alt til objekter og operatorer, viste programmet seg ubrukelig (Norvig 1992).

GPS og lignende programmer var altså basert på prinsippet om at programmets intelligens lå i måten det kom frem til løsninger på. En annen retning innenfor kunstig intelligens forsøkte derimot å lage problemløsningsprogrammer som var basert på kunnskap heller enn på resonneringsevne. Disse programmene virker på den måten at de har tilgang på kunnskap innenfor et gitt domene og bruker denne kunnskapen til å løse oppgaver innenfor dette området. Motsatt av de generelle problemløserne, som GPS, var disse kunnskapsbaserte systemene

altså spesifikke problemløsere som tok for seg problemer innenfor et helt konkret område.

Som et eksempel kan man ta Dendral, det første store prosjektet grunnlagt på dette prinsippet, påbegynt i 1965 (Buchanan et al. 2006). Med dette systemet ville man forsøke å etterligne de analytiske evnene til de få kjemikere som kan anslå molekylstrukturer i kjemiske substanser basert på data fra massespektrogrammer. Dette er et svært snevert definert problemfelt, en veldig spesialisert gren av kjemi. Samtidig er det også et langt mer komplisert problem enn de enkle, om enn generelle, problemene man satte GPS til å løse. Sistnevnte problemer er ikke bare enklere å løse, de har også ofte et fasitsvar. For eksempel kan man fort regne ut om et regnestykke er løst riktig eller ikke. Når det derimot gjelder anslag av molekylstrukturer kan flere konklusjoner passe bra med dataene fra massespektrogrammet og man kan ikke være sikker på at en konklusjon er riktig. Det finnes ingen måte å bevise at et anslag passer bedre enn et annet.

Dendrals etter hvert vellykkede anslag av molekylstrukturer førte til at man i større grad vektla kunnskap som en viktig del av kunstig intelligens og gikk vekk fra generelle og kunnskapsløse problemløsningsmodeller.

2.1 Konsept og visjoner

Dendral og lignende systemer som oppsto ut fra dette prosjektets suksess ble kalt for kunnskapsbaserte systemer eller ekspertsystemer. Disse to navnene brukes om hverandre for å benevne systemer av denne typen. Kunnskapsbaserte kalles slike systemer fordi de er basert på prinsippet at kunnskap er en viktig komponent i problemløsning. Derfor forsøker slike systemer å etterligne dem med mest kunnskap og erfaring, nemlig ekspertene. På denne måten blir også ekspertene de prestasjonsmessige idealene for ekspertsystemtradisjonen. Eksperters prestasjoner blir sett på som det beste menneskelig problemløsning har å tilby. Bak denne tanken ligger hele premisset for å utvikle ekespertsystemer. Men som vi skal se senere er ikke alle like enige i dette premisset.

Av ekspertsystemer forventer man at de løser problemer like bra som en ekspert fra det samme fagfeltet som kunnskapsbasen omhandler. I tillegg skal systemer av denne typen også kunne gjøre rede for hvordan de kommer frem til en løsning og de skal kunne begrunne hvorfor de mener denne løsningen er den riktige.

Hvorfor er det så interessant å etterligne eksperter? Jo mer man beveger seg bort fra allmenne kunnskapsområder, som for eksempel medisin, til spesialfelter under disse, som hjernekirurgi, jo færre blir nødvendigvis antallet mennesker som befatter seg med disse spesialfeltene. Eksperter er derfor mer sjeldne jo mer eksotisk deres spesialområde er. Innenfor disse spesialområdene kan eksperter være sterkt etterspurt av ulike aktører som selv ikke har noen kompetanse innenfor det aktuelle området. Dette kan være institusjoner, bedrifter eller privatpersoner som over kortere eller lengre tidsrom har bruk for spesialisters kunnskaper. For eksempel finnes det omtrent 5000 parfymeeksperter i verden (Kengpol & Wangananon 2006). For bedrifter som fremstiller parfymer er det imidlertid svært viktig å treffe ønskene til sine kundegrupper. For å finne ut hvilke typer lukter man skal bruke i sine produkter er parfymeindustrien derfor avhengige av disse 5000 ekspertene.

Denne etterspørselen etter enkelte ekspertkunnskaper fører til to ting. For

det første fører økt etterspørsel til at mange er villige til å betale mer for å få fatt på den kompetansen de trenger. Dersom et utall av parfymeproduserende selskaper skal konsultere totalt fem tusen eksperter mer eller mindre hyppig, sier det seg selv at prisen for en konsultasjon blir svært høy (Kengpol & Wangananon 2006). Slik kan det bli en kostbar affære å få eksperthjelp.

For det andre fører etterspørselen etter ekspertise til at de aktuelle ekspertene får mindre tid til andre oppgaver som samfunnsmessig sett kanskje er viktigere, slik som forskning eller utdannelse innenfor det aktuelle spesialområdet.

Den store fremtidsvisjonen når det gjaldt ekspertsystemer på 1970- og 1980-tallet, var derfor at de skulle kunne gjøre sjeldne eksperters ekspertise lettere tilgjengelig for alle som hadde bruk for den. Som et eksempel på at ekspertise blir gjort lettere tilgjengelig kan man tenke seg at et lite legekontor trenger råd fra en spesialist. Klinikken har ingen ekspert på det aktuelle området blant sine ansatte. En mulighet er da å kontakte en travel spesialist ved det nærmeste universitetssykehuset. Denne spesialisten er for sin del kanskje ikke spesielt interessert i å svare på eksterne henvendelser og ser dem muligens som en distraksjon fra sine egentlige arbeidsoppgaver. Enklere for alle parter ville det vært om legekontoret hadde et ekspertsystem for det aktuelle problemområdet. Kontoret hadde hatt ekspertisen tilgjengelig når som helst og eksperten på universitetssykehuset hadde sluppet stadige forstyrrelser.

Et annet eksempel gis av Duan, Fu, & Li (2002). Disse forfatterne beskriver et system der fiskeoppdrettere i Kina kan få hjelp til å diagnostisere sykdommer i sine fiskepopulasjoner for tidsnok å kunne treffe de tilsvarende mottiltakene. Siden oppdrettsanlegg ofte ligger lite sentralt til, overlever ikke mange fisk dersom de få ekspertene som finnes skal reise rundt til hver enkelt oppdretter å foreta diagnoser. Ved hjelp av ekspertsystemet kan en diagnose stilles på stedet og behandling påbegynnes øyeblikkelig.

2.2 Hvordan er et ekspertsystem bygget opp?

Et ekspertsystem består av minst to komponenter, nemlig en kunnskapsbase og en slutningsmekanisme (Buchanan et al. 2006). Kunnskapsbasen er, som navnet antyder, en mengde med kunnskap. Denne kunnskapen er kodifisert på et format som et program er i stand til å forstå. Kunnskapsbasen er kort og godt det systemet vet om sitt fag og det er denne kunnskapen som skal brukes til å løse brukernes problemer.

Slutningsmekanismen er selve motoren i et ekspertsystem. Dette er en algoritme som resonnerer over kunnskapen i kunnskapsbasen. Basert på den informasjonen brukeren av systemet gir, forsøker slutningsmekanismen å komme frem til en konklusjon gjennom kunnskapen systemet har. I tillegg til å finne konklusjoner må slutningsmekanismen også gjøre sin resonnering gjennomsiktig for brukeren. Brukeren skal kunne skjønne hvorfor systemet stiller de spørsmål det gjør og skal kunne følge systemets resonnement.

Jeg skal nå gå nærmere inn på hver av de delene som er nevnt ovenfor. Først ut er kunnskapsbasen.

2.2.1 Kunnskapsbasen

Kunnskapsbasen er det stedet der systemets kunnskap om et problemområde ligger. Denne kunnskapen har systemet felles med sine menneskelige forbilder, for kunnskapsbasen er ideelt sett en kopi av faktiske eksperters kunnskap. Prosessen med å overføre kunnskap fra ekspert til kunnskapsbase kalles for kunnskapsakvisisjon og er, som vi skal se, ikke uproblematisk.

(Buchanan et al. 2006) skiller mellom to typer kunnskap som eksperter er i besittelse av, nemlig faktisk og heuristisk kunnskap. Det er viktig å fange opp begge disse formene for kunnskap når man lager en kunnskapsbase.

Faktisk kunnskap er det man vanligvis forstår under kunnskap. Det er kunnskap som er allment akseptert som korrekt av fagfolk innenfor det aktuelle området og som finnes tilgjengelig i form av lærebøker og journaler. Strengt tatt trenger man som lekperson ikke å konsultere en ekspert for å bruke slik faktisk kunnskap da denne kunnskapen jo er offentlig tilgjengelig for alle som måtte interessere seg for den. I praksis er det imidlertid umulig for en lekperson å sette seg inn i faglitteratur for et eller annet spesialisert og komplisert fagområde. Eksperter har selv gjerne brukt årevis på å tilegne seg denne kunnskapen. Derfor er eksperter i praksis nødvendig dersom man må løse et problem som innebærer faktisk kunnskap, til tross for at denne kunnskapen er fysisk lett tilgjengelig.

Selv om man som lekperson mot alle odds skulle klare å sette seg inn i avansert faglitteratur er man fortsatt langt vekk fra å kunne løse et problem på samme nivå som en ekspert. Faktisk kunnskap er nemlig ikke nok til å etterligne en ekspert. Utover slik faktisk kunnskap er eksperter også i besittelse av en slags metakunnskap, en kunnskap om kunnskapen, om hvordan man best løser problemer innenfor deres fagfelt. Det er dette Buchanan et al. (2006) kaller for heuristisk kunnskap. Denne heuristiske kunnskapen representerer et større problem for utviklerne av kunnskapsbaser enn den faktiske kunnskapen.

I motsetning til den faktiske kunnskapen er heuristisk kunnskap individuell. Det er fordi den heuristiske kunnskapen har oppstått i eksperten som følge av dennes problemløsningserfaring. Slik har hver enkelt ekspert sin egen heuristiske kunnskap dannet på grunnlag av akkurat denne ekspertens erfaringer innenfor fagfeltet.

Siden heuristisk kunnskap er individuell er den sjelden diskutert i offentlige, faglige kanaler som journaler og finnes ikke lett tilgjengelig slik som den faktiske kunnskapen gjør. Det er allikevel svært viktig å få denne heuristiske kunnskapen med i et ekspertsystems kunnskapsbase dersom man skal ha et håp om å etterligne eksperten selv. Heuristisk kunnskap er jo en av de tingene som presumptivt skiller en ekspert fra en novise. Følgelig må en kunnskapsbase som skal brukes til å løse de samme problemene også ha slik heuristisk kunnskap.

En kunnskapsbase må altså inneholde både faktisk og heuristisk kunnskap dersom man skal ha et håp om å etterligne de ekspertene som man baserer kunnskapsbasen på. Hvordan får man fatt i denne kunnskapen fra eksperten og hvordan representerer man den i kunnskapsbasen?

2.2.2 Hvordan opprettes kunnskapsbasen?

Kunnskapsbasen lages som regel ved hjelp av en eller flere eksperter fra det fagområdet kunnskapsbasen skal lages for. Det er to grunner til dette. For det første trenger man ekspertens ovenfor nevnte individuelle heuristiske kunnskap

hvis man vil etterligne denne eksperten. For det andre forenkler eksperters deltakelse kunnskapens tilegnelse av faktisk kunnskap. Uten en ekspert kunne man muligens klare seg dersom problemområdet er så enkelt at utviklerne av kunnskapsbasen kan legge sin egen erfaring til grunn for kunnskapsbasen (Duan, Edwards & Xu 2005). Hvor interessant et slikt simpelt system er for allmennheten er imidlertid et åpent spørsmål. I forhold til alle relativt kompliserte problemområder er derfor bruken av en eller flere eksperter vanlig praksis under opprettelsen av kunnskapsbaser.

Ekspertene som deltar i systemutviklingen lager imidlertid ikke kunnskapsbasen alene. Dette vil sjelden fungere fordi det forutsetter at eksperten vet en hel del om de indre detaljene i ekspertsystemet. Derfor er opprettelsen av en kunnskapsbase oftest et samarbeid mellom ekspert og systemutvikler. Systemutvikleren kjenner systemet og vet hvordan kunnskap best bør representeres i dette, men har mer eller mindre ingen kjennskap til ekspertens fagfelt. Eksperten på sin side vet ikke hvordan systemet fungerer men kjenner sitt fag. Slik fungerer systemutvikleren som mellomledd mellom ekspert og system. I denne rollen blir systemutvikleren gjerne kalt for en kunnskapsingeniør og har som en av sine oppgaver å sørge for at den kunnskapen eksperten bidrar med blir representert og organisert på en slik måte at den kan brukes i systemet (Carroll 1987).

En vanlig måte å gjøre denne kunnskapsrepresentasjonen på består i å formulere ekspertens kunnskap i regler. Disse reglene har som oftest en nokså enkel struktur. De består gjerne av en premissdel og en konklusjonsdel. Premissdelen inneholder en eller flere premisser. Hvis alle disse premissene slår til vil konklusjonen, som befinner seg i konklusjonsdelen av regelen, være riktig.

Som et eksempel på en slik regel kan man ta en forenklet utgave av en av Mycins regler (Norvig 1992): Hvis bakteriekulturen er funnet i blodet og hvis pasientens forbrenningsskader er alvorlige, så er identiteten til organismen pseudomonas. Denne regelen omhandler hva slags organisme som forårsaker en sykdom hos en pasient og bruken av regelen er rett frem. Hvis legen fant bakteriekulturen der organismen lever i en blodprøve og hvis pasienten kan sies å ha alvorlige brannskader, så er organismen pseudomonas. Hvis en eller begge premissene ikke stemmer, det vil hvis legen ikke fant bakteriekulturen i blodet eller at pasienten ikke er alvorlig brent, så er denne regelen ugyldig og man kan i stedet prøve en annen regel.

Før man kan lage regler som reflekterer ekspertkunnskap må man imidlertid først få tak i denne kunnskapen. Den tradisjonelle tilnærmingen til dette har vært å intervjue eksperten. Men det byr på en del vanskeligheter. For det første er deler av ekspertens kunnskap så innarbeidet at eksperten har vanskeligheter med å artikulere denne kunnskapen. For det andre er ekspertene som regel vant til å diskutere sine fag i mer eller mindre esoteriske fagspråk som legfolk kan ha store problemer med å forstå. Derfor er det heller ikke lett for systemutvikleren å sette seg såpass inn i fagområde at hun kan føre en fornuftig samtale med eksperten. Vi kommer til å se nærmere på problematikken rundt opprettelsen av kunnskapsbasen i kapittel fire.

Ikke bare er det problematisk å hente kunnskapen fra eksperten. Det er heller ikke lett å organisere denne kunnskapen slik at den kan brukes av det ferdige systemet. Noen regler kan for eksempel omhandle identifiseringen av en bakterie, slik som eksempelregelen ovenfor. Andre regler kan ha den hensikt å bestemme en bakteries form. Den sistnevnte regelen kan brukes til å komme nærmere en løsning i forhold til den førstnevnte regelen. Hvis man vet at en bakterie er

stavformet, og dette faktum oppfyller et premiss i den førstnevnte regelen, så er man litt nærmere å kunne konkludere med den første regelens konklusjon. Noen reglers premisser kan altså besvares ved å følge regler som så og si har en lavere orden i den forstand at de kan løse underproblemer til høyereordens regler. Denne strukturen i kunnskapsbasen må stemme overens.

Hver regel må også være en unik størrelse. Det vil si at ingen regler må overlappe. Flere regler kan godt konkludere til den samme konklusjonen, men da gjennom ulike premisser. Hver enkelt regel må ta seg av en liten bit av hele kunnskapsbasen og må derfor være enestående. Derfor må kunnskapsbasens konsistens overprøves regelmessig slik at det ikke sniker seg inn regler som for eksempel oppnår samme konklusjon gjennom en eller flere av de samme premissene. Denne konsistenssjekkingen er en del av arbeidet med kunnskapsbasen så lenge man legger til nye regler i denne.

En vanlig strategi for å lage en kunnskapsbase er å begynne enkelt med noen få regler. Disse kan så overprøves av eksperten og eventuelt endres dersom denne har noe å utsette på dem eller hvis de åpenlyst kommer frem til galt resultat. Så kan man sakte utvide kunnskapsbasen med nye regler mens man hele tiden passer på at de ikke ødelegger for reglene som allerede finnes. Eventuelt kan det bli nødvendig å endre på allerede eksisterende regler underveis ettersom nye regler kommer for dagen. Denne inkrementelle måten å utvikle kunnskapsbasen på forutsetter at det allerede finnes en slutningsmekanisme som bruker reglene slik at man kan kjøre systemet og se hva slags konklusjoner det gir.

2.2.3 Slutningsmekanismen

Slutningsmekanismen er den andre helt nødvendige delen av et ekspertsystem. Mens kunnskapsbasen består av passiv kunnskap organisert i regler er slutningsmekanismen den aktive, arbeidende delen av ekspertsystemet. Den styrer hvilke regler fra kunnskapsbasen som brukes og forsøker å tilfredsstille premissene i hver regel etter tur. Hvis en regel viser seg å være gal, det vil si at et eller flere premisser ikke slår til, går slutningsmekanismen videre til neste relevante regel.

En vanlig måte å gå gjennom kunnskapsbasens regler på er gjennom en teknikk som kalles baklengs kjeding (Buchanan et al. 2006). Baklengs kjeding tar utgangspunkt i et mål som man vil oppnå og arbeider videre ut fra dette. Hva målet kan være kommer an på hvilket kunnskapsområde som ekspertsystemet omhandler. I et Mycin-lignende system kan et mål for eksempel være å bestemme identiteten til en organisme i en bakteriekultur. Slutningsmekanismen vil da finne frem til alle regler som er relevante for denne problemstillingen. Dette vil være regler som slutter til målet. Det vil si at de har konklusjoner som omhandler organismers identitet. Eksempelregelen fra lenger opp i kapittelet vil i dette tilfellet være en relevant regel. Den konkluderer med en organismes identitet, pseudomonas.

Den baklengse kjedingen går ut på at hvert premiss i en regel behandles som et delmål. For eksempel: Sett at man har et ekspertsystem som skal hjelpe til med å artsbestemme fugler. Artsbestemming vil da være hovedproblemet som systemet skal løse. Men for å kunne løse dette problemet må systemet også kunne løse en hel del underproblemer som for eksempel å bestemme form på nebb og vinger. Dersom brukeren oppgir at hun vil bestemme artstilhørigheten til en fugl vil systemet finne frem til alle regler som konkluderer med en

artsbestemmelse. Dette vil være regler som gir konklusjoner som for eksempel spurv eller gråmeis. Slutningsmekanismen vil så arbeide videre med disse reglene. Hvis slutningsmekanismen, for å løse artstilhørighetsproblemet, møter på et underproblem, for eksempel formen på fuglens nebb, vil den så og si bevege seg et hakk ned i regelhierarkiet og begynne på nytt igjen. Det vil si at slutningsmekanismen finner alle reglene som konkluderer med en nebbform. Når dette problemet er løst fortsetter slutningsmekanismen der den slapp hovedproblemet. Eventuelt må det først løses et underproblem av underproblemet. Siden flere nivåer av underproblemer muligens må løses før man kan løse hovedproblemet, er det slutningsmekanismens oppgave å holde styr på hvor i løsningshierarkiet man befinner seg. Slik baklengs kjeding egner seg godt til domener der man kjenner målet man vil oppnå, men ikke kjenner veien til målet.

For å tilfredsstille reglers premisser kan slutningsmekanismen enten spørre brukeren eller bruke kunnskapsbasen. I eksempelet ovenfor kan slutningsmekanismen enten spørre brukeren direkte om formen på nebbet eller den kan finne frem alle reglene som konkluderer på nebbform og forsøke å løse disse reglene først. Som regel vil slutningsmekanismer først prøve å løse problemet selv og først spørre brukeren om verdien til et premiss når det selv ikke kommer noen vei. Senest når kunnskapsbasen ikke inneholder noen flere regler lenger ned i regelhierarkiet må slutningsmekanismen spørre brukeren.

2.2.4 Håndtering av usikkerhet

Det kan ofte være usikkerhet knyttet til en konklusjon. For eksempel kan et gitt symptommønster tyde på at en sykdom er den mest sannsynlige uten at andre sykdommer av den grunn kan utelukkes fullstendig. Ekspertsystemer må derfor ofte kunne håndtere usikkerhet. Dette gjøres gjerne ved at reglenes konklusjonsdel har en såkalt sikkerhetsfaktor knyttet til seg. Dette vil være en tallverdi som sier noe om hvor sikker man er på en regels konklusjon dersom regelens premisser stemmer. Denne verdien kan modifiseres videre dersom premissene også er usikre. Slike sikkerhetsfaktorer oppgis for hver regel av den eller de eksperter som systemet får sin kunnskap fra og ligger typisk mellom -1, som tilsier at noe helt sikkert ikke forekommer, og 1 som tilsier at noe forekommer helt sikkert. Gjennom regneoperasjoner på sikkerhetsfaktorer får man et visst mål på hvor sannsynlig systemet mener en konklusjon er.

Som et eksempel kan eksempelregelen ovenfor utvides med en sikkerhetsfaktor. Hvis bakteriekulturen er funnet i blodet og hvis pasientens forbrenningsskader er alvorlige, så er identiteten til organismen pseudomonas med en sikkerhetsfaktor på 0,7. I dette tilfellet har eksperten altså vært nokså sikker på at det dreier seg om pseudomonas dersom denne regelens premisser er oppfylt.

Sikkerhetsfaktorer gjør imidlertid at slutningsmekanismen blir langt mer komplisert. Den må i tillegg til å manipulere regler nå også kombinere ulike sikkerhetsfaktorer. Hvis to regler ad ulike veier konkluderer med den samme konklusjon må sikkerhetsfaktoren til den endelige konklusjonen beregnes på grunnlag av sikkerhetsfaktorene til begge reglene.

Sett at en regel konkluderer med en konklusjon med sikkerhet på 0,60 og en annen regel konkluderer med den samme konklusjonen men med en sikkerhet på 0,40. Man kan i dette tilfellet ikke servere brukeren med to ulike sikkerhetsvurderinger for en og samme konklusjon. I denne situasjonen

ville Mycin summert de to sikkerhetsfaktorene og fra denne summen trukket fra produktet av de samme sikkerhetsfaktorene. Altså 0,60 + 0,40 - (0,60 * 0,40). Svaret på dette regnestykket er 0,76 som Mycin ville ha oppgitt som sikkerhetsfaktor for den endelige konklusjonen. Ulike metoder som dette brukes for å beregne sikkerhetsfaktorer i ulike regelkombinasjoner. Slutningsmekanismen kan slik alltid bidra med en sikkerhetsfaktor for sin konklusjon. Som vi skal se i kapittel tre er sikkerhetsfaktorer imidlertid ikke det samme som sannsynligheter.

2.2.5 Gjennomsiktighet

Det er viktig for brukerens tillit til systemet at dette er i stand til å begrunne hvorfor det oppfører seg som det gjør. Brukeren må til en hver tid kunne forstå hvordan systemet resonnerer. Hvis ikke dette er tilfellet kan ikke brukeren reagere dersom vedkommende ser en brist i systemets resonnement. Dette er ikke bare viktig i forhold til sluttbrukerne av systemet, men også for utviklingen av kunnskapsbasen. Eksperter kan slik kjøre systemet og se om systemet oppfører seg slik de forventer. Derfor må en bruker kunne spørre systemet om hvorfor det stiller de spørsmål det gjør og om hvordan det kom frem til en eventuell konklusjon.

På grunn av dette begrunnelseskravet må slutningsmekanismen også overvåke hvilke regler den har vurdert som ugyldige eller gyldige, hvilken regel den holder på med i øyeblikket og hvilket problem systemet prøver å løse på et gitt punkt i en programmkjøring. Dersom en bruker vil vite hvorfor systemet spør om en ting, vil systemet svare med å henvise til den regelen den i øyeblikket forsøker å bekrefte. Som et eksempel kan man ta en enkel Mycin variant beskrevet i Norvig (1992). Her vil brukeren vite hvorfor systemet spør etter en organismes aerobisitet. (Eksempelet er lettere endret og oversatt av forfatteren.)

```
Ekspertsystem: Hva er AEROBISITETEN til ORGANISME-1?
Brukeren taster inn: Hvorfor
Ekspertsystem: Det er kjent at:
   1) Organismens gram er neg
   2) Organismens morfologi er rod
       Derfor,
          Regel 107:
             Hvis
             1) Organismens aerobisitet er aerobisk
                Så er det bevis (0,8) for at
             1) Identiteten til organismen er
                enterobacteriaceae.
```

Dersom brukeren ber om en begrunnelse for en konklusjon vil systemet vise frem rekken av gyldige regler det har kjedet sammen i løpet av kjøringen.

I de tidlige ekspertystemene ble de delene som er blitt beskrevet hittil, kunnskapsbasen og slutningsmekanismen, bygget opp slik at de var sammenvevd. Det vil si at slutningsmekanismen var spesielt utviklet bare for å kunne håndtere akkurat sin kunnskapsbase. Etter hvert begynte man å skille ut generelle komponenter, som slutningsmekanismen i egne programmer, såkalte ekspertsystemskall.

2.3 Ekspertsystemskall

Man fant fort ut at noen av de samme teknikkene stadig ble brukt i ekspertsystemer. Slutningsmekanismen er naturligvis et meget godt eksempel på dette. Intet ekspertsystem kan klare seg uten. De store forskjellene mellom systemene lå altså den ulike kunnskapen i systemenes kunnskapsbaser. Men siden det ikke finnes så veldig mange måter å representere kunnskap på, var kunnskapsbasene nokså like i oppbygning. Derfor var det et kort steg å begynne å utvikle slutningsmekanismer som var uavhengige av kunnskapsbasen. Det vil si at de var generelle heller enn spesielt utviklet for å kunne fungere opp mot en konkret kunnskapsbase. Slik kunne man samle hele «infrastrukturen» rundt prosessen med å trekke slutninger ut fra en regelmengde i et uavhengig system. Siden man kan tenke seg at slike systemer er bygget rundt kunnskapsbasen som et skall, ble de kalt for ekspertsystemskall.

Et ekspertsystemskall er slik en mer generell problemløser som minner litt mer om programmer som GPS. Disse generelle skallene kan så bruke kunnskapsbaser som representerer kunnskap innenfor et spesielt område. Slik kan et ekspertsystemskall også håndtere ulike kunnksapsbaser så lenge disse er på formatet skallet er laget for å håndtere. Et eksempel på et slikt skall er Emycin, en forkortelse for Essential Mycin, som ble utviklet ut fra ekspertsystemet Mycin. Emycin er et skall som kan resonnere over kunnskap og forklare resultatene det kommer frem til, uavhengig av spesifikk kunnskap. I eldre systemer var slutnigsmekanismen skreddersydd for den aktuelle regelbasen. En ny regelbase innebærer da at en ny slutningsmekanisme må lages. Ved hjelp av ekspertsystemskall kan man slik lette utviklingen av ekspertsystemer ved å bruke allerede eksisterende skall.

2.4 Sammenfatning

Dette kapittelet har tatt for seg ekspertsystemer i sin alminnelighet. Vi har sett at ekspertsystemidéen oppstod innenfor forskningsretningen kunstig intelligens og at ekspertsystemene var spsifikke problemløsningsprogrammer som skulle kunne løse problemer innenfor et gitt fagområde ved hjelp av kunnskap fra dette området. Interessen for slike kunnskapsbaserte systemer tok seg opp etter at generelle problemløsningsprogrammer ikke klarte å løse annet enn enkle problemer.

Vi så videre at ekspertsystemfeltet forsøkte å emulere menneskelige eksperter for å gjøre disses spesialistkunnskap lettere tilgjengelig for allmennheten. Ved å gjøre dette gjorde man også ekspertene til prestasjonsmessige idealer for ekspertsystemene.

Når det gjaldt ekspertsystemers oppbyggning så vi at denne består av minimum en kunnskapsbase og en slutningsmekanisme.

Kunnskapsbasen er stedet der systemets kunnskap lagres. Kunnskapen i kunnskapsbasen med henter man fra eksperter innen det aktuelle fagområdet som kunnskapsbasen omhandler. Kunnskapsbasen utvikles derfor oftest gjennom et sammarbeid mellom systemutvikler og ekspert. Eksperten bidrar med sin fagkunnskap og systemutvikleren represeneterer denne kunnskapen i basen. Ekspertens kunnskap representeres vanligvis ved hjelp av regler bestående av et sett med premisser og en konklusjon.

Det ble også sagt at det er problematisk å få tak i ekspertens kunnskap.

Dette skyldes at deler av ekspertens kunnskap kan være ubevisst. Det er også gjerne vanskelig for eksperten å kommunisere sin kunnskap til en uvitende systemutvikler.

Vi så at slutningsmekanismen er den delen av ekspertsystemet som bruker kunnskapen fra kunnskapsbasen for å løse problemer. Slutningsmekanismen går gjennom regler relevante for den aktuelle problemstillingen og forsøker å bekrefte disse. Vi så et en vanlig strategi for å bevege seg gjennom reglene på er ved baklengs kjeding der hvert premiss i en regel behandles som et delmål på veien til å løse hovedproblemet

Det ble påpekt at ekspertsystemer også ofte må hanskes med usikker informasjon. Dette blir gjerne gjort ved at konklusjonene i kunnskapsbasens regler får en sikkerhetsfaktor. Det er slutningsmekanismens jobb å beregne endelige sikkerhetsfaktorer for konklusjoner den presenterer.

Slutningsmekanismen bør også være gjennomsiktig slik at brukeren kan følge dens resonnement. Derfor må slutningsmekanismen kunne begrunne både spørsmål til brukeren og konklusjoner.

Tidlige ekspertsystemer var preget av at slutningsmekanismen var spesiellt laget for å bruke akkurat sin kunnskapsbase. Etterhvert utviklet man derimot ekspertsystemskall. Disse består av mer generelle slutningsmekanismer som kan bruke forskjellige kunnskapsbaser så lenge de er i stand til å forstå reglene i den aktuelle kunnskapsbasen.

Neste kapittel handler om hvilke problemer som ekspertsystemene slet med på 1980-tallet.

3 Problemer med ekspertsystemer på 1980-tallet

Etter utviklingen av flere vellykkede ekspertsystemer på 1960 og 1970 tallet rådet det stor optimisme blant dem som trodde at ekspertsystemer kunne komme til å erstatte menneskelig problemløsning. Tidlige forsøksmessige systemer som Dendral, Mycin og senere kommersielle systemer som Prospector og R1/Xcon, gjorde at den store visjonen om å erstatte menneskelige eksperter med maskiner rykket nærmere en realisering. Men denne realiseringen lot vente på seg og i dag er begrepet blandet suksess en karakteristikk som ofte brukes i forbindelse med ekspertsystemfeltet på slutten av 1980-tallet.

Betegnelsen blandet suksess innebærer imidlertid at noen systemer var vellykkede mens andre ikke var det. Durkin (1996) gjør en gjennomgang av ekspertsystemene på 1980-tallet og tidlig på 1990-tallet. Han rapporterer at de fleste ekspertsystemer laget i denne perioden er å finne innenfor tre hovedområder, nemlig medisin, produksjon og forretningsdrift. I forhold til disse tre områdene finner han de mest vellykkede systemene innenfor produksjon og forretningsdrift, altså sektorer dominert av private selskaper. Durkin, kaller systemene som orienterte seg mot disse sektorene for kommersielle systemer. Dette er ekspertsystemer som gjør økonomiske foretak mer lønnsomme.

Når det gjelder de direkte arvtagerne til Mycin, medisinske ekspertsystemer, levde ikke disse systemene opp til de høye forventningene enkelte hadde til dem. Dette var ekspertsystemer som skulle hjelpe medisinsk personale med ulike oppgaver, som regel diagnostisering innenfor spesialfelt. Heathfield (1999), som beskjeftiget seg med medisinske ekspertsystemer på denne tiden, klager over at ikke et eneste av de systemene hun var med på å utvikle ble tatt i rutinemessing bruk. Man kom aldri lenger enn at man fikk utplassert prototyper av systemene for å teste disse. Også Simpson, Kingston & Molony (1999) påpeker at bare et fåtall intelligente medisinske systemer er blitt tatt i bruk.

Medisinske ekspertsystemer ser altså ut til å ha vært mer problematiske enn kommersielle systemer. Vi tar først for oss mulige grunner til dette. Deretter ser vi på de problemene som alle ekspertsystemer på denne tiden hadde, enten de var medisinske eller kommersielle. Den siste delen av kapittelet handler om et stort problem for spesielt medisinske systemer nemlig håndtering av usikkerhet.

3.1 Kommersielle versus medisinske ekspertsystemer

Som det ble sagt innledningsvis var suksessen til ekspertsystemfeltet blandet på 1980-tallet. Noen kommersielle systemer innenfor foretningsdrift og produksjon ble tatt i bruk mens medisinske systemer ikke klarte å slå gjennom på samme måte. Denne ulike evnen til å hevde seg i bruk har flere grunner.

Når det gjelder de kommersielle systemene var heller ikke disse noen stor suksess i starten. Dette skyltes i første rekke at de første kommersielle systemene var for ambisiøse (Durkin 1996). Suksesshistorier som Prospector, et geologisk ekspertsystem som hjalp til med å finne en malmforekomst verd hundre millioner dollar, inspirerte systemutviklerne til å forsøke å nå nye horisonter med sine ekspertsystemer. Man prøvde å sprenge grenser ved å lage systemer som skulle løse problemer som selv de beste eksperter ikke kunne løse. Dette klarte man ikke. Mange av de fullførte systemene ble skuffelser i forhold til de høye forhåpningene man hadde hatt og de store løftene man hadde gitt brukerne.

Kommersielle ekspertsystemer slo først an når man begynte å utvikle

enklere systemer som konsentrerte seg om helt konkrete, veldefinerte, noen ganger også trivielle, oppgaver. Durkin erkjenner at disse systemene nok ikke imponerte forskere fra kunstig intelligens tradisjonen og heller representerte skuffelser i forhold til de store visjonene fra begynnelsen av det samme tiåret. Disse relativt enkle systemene var imidlertid i stand til å gjøre seg gjeldende i forretningssammenheng. Et eksempel som trekkes frem av Durkin er EvEnt, et ekspertsystem som skulle hjelpe den franske banken Evalog med å vurdere lånesøknader. Dette systemet frigjorde bankansatte fra gjentagende, rutinemessige gjennomganger av lånesøknadene. EvEnt gjorde at kostnadene forbundet med vurderinger av lånekunder gikk ned til en tidel, gjorde at banken kunne vurdere flere lånesøkere og reduserte dessuten bankens risiko ved å låne ut penger.

Kommersielle systemer var også lettere å ta i bruk fordi disse systemene var nyttige selv om de ikke gav perfekte eller optimale løsninger hele tiden (Schwartz & R. S. Patil 1987). Om et ekspertsystem som skal hjelpe til med å konfigurere datamaskiner, slik som X1, av og til gjør en feil er ikke så viktig. Dette kan man lettere akseptere enn en feil diagnose i forhold til en svært syk pasient.

Heathfield (1999) peker på noen årsaker til at medisinske ekspertsystemer ikke ble brukt i noe særlig omfang. Som det aller største hinderet ser hun ingen teknisk årsak men derimot de negative holdningene til personalet som skulle bruke systemene. Hun mener at leger var svært skeptiske til medisinske ekspertsystemer. De fryktet at deres ferdigheter, som det nok er knyttet betydelig profesjonell stolthet til, skulle bli erstattet av intelligent programvare. Hvis et ekspertsystem skulle vise seg å være svært nøyaktig, fryktet legestanden kanskje å bli pålagt å alltid konsultere et ekspertsystem og slik bli marginalisert som yrkesgruppe. Slik sett var kanskje selve motivasjonen til å ta i bruk ekspertsystemteknologi høyere innenfor private selskaper som så en mulighet for større profitt.

En annen årsak til at medisinske ekspertsystemer ikke slo gjennom med forventet styrke har sin bakgrunn i at mange av disse systemene ble utviklet av forskningsinstitusjoner som mer brukte fagfeltet medisin som et interessant problemområde for å prøve ut nye teorier eller programmeringsteknikker. Å lage fungerende medisinske systemer var mindre prioritert blant disse utviklerne (Heathfield 1999). Muligens interesserte man seg også for medisin fordi dette samfunnsmessig viktige og prestisjefylte området gav muligheter for større forskningsmidler. Denne holdningen førte til at systemene ble basert på svært overfladiske og dårlig gjennomførte brukerundersøkelser. Dette ledet igjen til systemer som ikke var tilpasset måten leger eller annet helsepersonal faktisk jobber på. Heathfield (1999) gir selv et eksempel på et slikt misforhold mellom arbeidsmåte og system. I det aktuelle problemområdet systemet ble utviklet for, innebar ekspertens arbeidsoppgave at hun studerte detaljer gjennom et mikroskop. Denne aktiviteten var ikke integrert i systemet. Dette innebar at brukeren av systemet hele tiden måtte skifte mellom mikroskopet og skjermen til datamaskinen systemet var installert på. En bedre løsning hadde vært om bildet fra mikroskopet kunne bli vist på den samme skjermen som en integrert del av brukergrensesnittet til systemet.

En tredje grunn til medisinske systemers fallitt, mener Heathfield (1999) var deres manglende evne til å håndtere usikkerhet. Dette er et spesielt prekært problem innen medisin. De fleste medisinske ekspertsystemer er tenkt å hjelpe til med diagnostisering men, som vi skal komme tilbake til senere,

er det medisinske domenet så komplisert at det svært ofte vil være knyttet usikkerhet til en diagnose. Diagnoser kan slik være fremsatt med større eller mindre grad av tiltro til at det dreier seg om den riktige diagnosen. Derfor er det viktig at ekspertsystemer er i stand til å gi sikkerhetsanslag i forhold til diagnoser de kommer frem til. Som nevnt i kapittel to har man forsøkt å bygge inn usikkerhetshåndtering i ekspertsystemer ved å bruke tallverdier, kalt sikkerhetsfaktorer, som uttrykker hvor sikkert systemet er på en gitt konklusjon.

Denne usikkerhetshåndteringen er imidlertid problematisk fordi systemets sikkerhetsanslag i bunn og grunn kommer fra eksperten. Eksperter er imidlertid svært dårlige til å gjøre slike sikkerhetsanslag (Brehmer 1980) og derfor er hele sikkerhetshåndteringen skjev fra begynnelsen av. Vi skal gå grundigere inn på denne problemstillingen senere i dette kapittelet. Vi venter imidlertid litt med dette fordi dette er et problem som spesielt treffer de medisinske ekspertsystemene. Ekspertsystemene på 1980-tallet var imidlertid plaget av en del generelle problemer enten deres fagområde var medisinsk eller ikke. Vi ser først nærmere på disse allmenne problemene.

3.2 Generelle problemer med 1980-tallets ekspertsystemer

3.2.1 Opprettelsen av kunnskapsbasen

Som det ble nevnt i kapittel to er det problematisk å opprette et ekspertsystems kunnskapsbase. Et problem er å få eksperten til å fortelle hvordan hun løser problemer. Dette er vanskelig fordi eksperter sjelden er vant til å diskutere sine resonnementer (Buchanan & Shortliffe 1984) og fordi mye av ekspertens kunnskap er så innarbeidet at den ofte blir brukt ubevisst. Sagt på en annen måte er eksperten seg ikke alltid bevisst de enkelte resonneringstrinnene hun bruker for å komme frem til en gitt løsning. For eksperten kan det virke som om hun løser enkelte problemer intuitivt, nærmest på grunnlag av en slags mageføelse. Det er spesielt den heuristiske kunnskapen som er ubevisst på denne måten.

Siden deler av ekspertens kunnskap kan være ubevisst kan det være vanskelig for eksperten å redegjøre for sin egen kunnskap eller fremgangsmåte. Derfor er det viktig at systemutvikleren hjelper eksperten med å gjøre denne kunnskapen eksplisitt. Systemutvikleren må få eksperten til å tenke over hva og hvorfor vedkommende gjør det hun gjør under løsningen av en gitt problemstilling. Ulike former for intervjuer, observasjoner av hvordan eksperter løser problemer eller analyser av hvordan eksperter løser testtilfeller kan hjelpe til med å kartlegge ekspertens kunnskap.

At eksperten har vanskelig for å artikulere sine egne resonneringstrinn er imidlertid ikke det eneste problemet med denne «melkingen» av ekspertens kunnskap. Ekspertene og kunnskapsingeniørene kommer som regel fra to forskjellige verdener. Ekspertene er spesialister som har detaljerte kunnskaper om et komplisert tema. Å kunne snakke om dette spesialiserte temaet krever som oftest kjennskap til et komplisert fagspråk. Dette fagspråket fordrer igjen at man har god kjennskap til den generelle disiplinen som ekspertens spesialisering er en del av. For eksempel må man, for å kunne følge en samtale mellom to hjernekirurger, også kunne mye om allmennmedisin i tillegg til hjernekirurgi. Siden systemutvikleren i regelen ikke har noe kjennskap

til ekspertens fagområde, foreligger det et kommunikasjonsproblem mellom systemutvikler og ekspert. Enkle ordforråd vil oftest være utilstrekkelige til å formidle kompleksiteten i eksperters kunnskap. Hvordan skal eksperten forklare kompliserte problemstillinger for en systemutvikler, som kan lite eller ingen ting om det aktuelle faget, på en slik måte at sistnevnte kan lage brukbare regler av dem?

Dette problemet blir noe mindre etter hvert som kunnskapsbasen utvikles. Systemutvikleren forstår sakte men sikkert mer av ekspertens fagspråk og blir derfor i større grad i stand til å stille eksperten fornuftige og relevante spørsmål. Eksperten og systemutvikleren klarer derfor etter hvert å samarbeide bedre og prosessen med å overføre ekspertens kunnskap til kunnskapsbasen går raskere (Buchanan & Shortliffe 1984).

På grunn av disse problemene med kunnskapsakvisisjonen er det et svært tidkrevende arbeid å omdanne eksperters kunnskap til regler i en kunnskapsbase. Opprettelsen av kunnskapsbasen regnes derfor som den vanskeligste og mest tidkrevende delen av arbeidet med å lage et ekspertsystem. For å få en idé om hvor lang tid dette kan ta, kan man se på ekspertsystemet Caduceus. Dette systemet, påbegynt på 1970-tallet, var et medisinsk ekspertsystem som omhandlet indremedisin. Eksperten som dette systemet ble bygget på brukte tre timer hver uke over seks måneder for å hjelpe systemutviklerne med å bygge skjelettet til dette systemet (Carroll 1987). Hele systemet var ikke ferdig før på midten av 1980-tallet.

Siden det er så vanskelig å overføre eksperters kunnskap til kunnskapsbaser blir kunnskapsbasene i regelen svært smale (Buchanan et al. 2006). Det vil si at man konsentrerer seg om å finne akkurat den kunnskapen som er kritisk for det aktuelle fagområdet og ser bort i fra mer generell kunnskap.

3.2.2 Tilgjengelighet

Som vi har sett lå mye av motivasjonen bak utviklingen av ekspertsystemer i ønsket om å gjøre eksperters kunnskap lettere tilgjengelig. Hvis dette skal kunne gjøres må naturlig nok også ekspertsystemet selv være tilgjengelig for eventuelle brukere. Derfor var det et problem at 1980-tallets ekspertsystemer generellt var vanskelig tilgjengelige. Det vil si at det kunne være tungvint for brukere å bruke systemet på det sted eller den tid de trengte det. Dette skyldtes at brukere var tvunget til å oppsøke ekspertsystemer på de maskinener de var installert på (Grove 2000, Duan et al. 2005). Dette er naturlig nok problematisk hvis brukeren må befinne seg på et bestemt sted for å løse et problem mens ekspertsystemet som kan hjelpe vedkommende med dette problemet bare er tilgjengelig på en datamaskin som står et helt annet sted.

Et eksempel tatt fra Grove (2000) er allmennlegen som er på husbesøk og trenger spesialisthjelp. Det er svært upraktisk for vedkommende å måtte reise tilbake til sitt kontor hvor et ekspertsystem er installert på en datamaskin. Eller sett at man har installert et medisinsk ekspertsystem i en sykehusavdeling. Systemet er tilgjengelig på en terminal et sted i avdelingen. Den vakthavende lege på inspeksjon må da allikevel bevege seg fra det rommet den aktuelle pasienten ligger på til stedet der terminalen står. Selv om dette muligens ikke er en veldig lang strekning høyner det allikevel terskelen for å bruke systemet. Det skal ikke høye terskelen til før mange brukere heller lar være å bruke et system (Grove 2000).

3.2.3 Oppdatering

Et annet problem på 1980-tallet var oppdatering av kunnskapsbasen. De fleste større systemer må oppdateres med jevne mellomrom. Dette gjelder spesielt for kunnskapsbaser siden ny viten kan oppdages eller man kan oppdage svakheter eller feil ved kunnskapsbasen. For eksempel kan man komme frem til nye tester, teorier, diagnoser eller behandlinger som igjen krever at nye regler må legges til i systemet eller at gamle regler må endres. Et eksempel som nevnes av McCarthy (1984) er ekspertsystemet Mycin og sykdommen Legionella. Denne sykdommen ble først forstått etter at Mycin var ferdig. Dersom Mycin skulle kunne hjelpe til med å diagnostisere Legionella måtte systemet i så fall først bli informert om denne sykdommen i form av nye regler.

Ekspertsystemers avhengighet av fungerende og oppdaterte kunnskapsbaser gjør at ekspertsystemer ikke er programmer man kan gjøre seg helt ferdig med, for så å sende det til brukerne. Utvikling og forbedring av kunnskapsbasen er en kontinuerlig prosess som pågår så lenge ekspertsystemet er i drift (Davis 1982, Buchanan & Shortliffe 1984). McCarthy (1984) gir et eksempel som tydelig viser at det å holde kunnskapsbasen oppdatert var et problem på 1980-tallet. Han spurte en del klinikere om det ville være passende å gi ekspertsystemet Mycin til leger som var villige til å bruke dette systemet slik at disse kunne installere Mycin på sine egne datamaskiner. En del av de spurte svarte da at dette ville være i orden dersom det fantes en måte å holde Mycins regelbase oppdatert i forhold til nye funn innenfor fagområdet.

Hvis nå et ekspertsystems kunnskapsbase skulle oppdateres med ny informasjon innebar dette på 1980-tallet at den nye oppdateringen, lagret på en diskett, måtte sendes per post til alle brukerne av systemet (Grove 2000, Duan et al. 2005). Selve installasjonen av oppdateringene kunne også være så komplisert at man ikke kunne anta at alle brukere ville få til dette selv (Duan et al. 2005). I tillegg innebar postforsendelse at man måtte holde styr på hvilke brukere som hadde fått tilsendt oppdateringer og hvilke brukere som enda ikke hadde fått noe (Grove 2000).

Oppdatering av kunnskapsbasen er helt avgjørende dersom et ekspertsystem skal kunne fungere etter den hele tiden gjeldende kunnskapen. Men denne oppdateringen av kunnskapsbasen var altså problematisk på 1980-tallet grunnet tungvinte postforsendelser og mye bokholderi.

3.2.4 Ekspertsystemers manglende robusthet

Enda et problem med 1980-tallets ekspertsystemer var deres manglende robusthet. Dette viser seg ved at ekspertsystemer faller voldsomt i ytelse når de må forsøke å løse problemer som muligens ligger innenfor det problemområdet som systemet er laget for, men som systemet allikevel ikke er forberedt på. En bruker kan jo prøve å bruke et system på en måte som ikke er tiltenkt av systemutviklerne. Dette gjelder gjerne veldig spesielle tilfeller eller unntak.

I slike tilfeller faller ekspertsystemers ytelser veldig raskt ned på et svært lavt nivå. Dette er rett og slett fordi systemet mangler de tilsvarende reglene for å kunne håndtere det aktuelle problemet. Det finnes i ekspertsystemer ingen gradvis overgang fra problemer som det finnes regler for og til problemer som det ikke finnes regler for. Uten regler ingen ytelse.

Også menneskelige eksperter vil være mindre nøyaktige når de skal løse

problemer som de ikke kan så mye om, men de vil allikevel kunne opprettholde en viss grad av nøyaktighet (Carroll 1987). Eksperter kan alltids litt om emner som ikke har direkte med deres spesialfelt å gjøre. De har allmenn kunnskap, ervervet gjennom skolegang og livserfaring, som dataprogrammer ikke har. Denne allmenne kunnskapen kan de bruke til å gjøre slutninger i tilfeller der deres spesialistkunnskap ikke strekker til. I forhold til ekspertsystemene har menneskelige eksperter slik en mykere overgang i ytelse fra problemer som de behersker til problemer som de ikke behersker. Ekspertsystemer har ingen slik generell kunnskap. Deres kunnskapsbaser er jo i regelen smale og fokusert på akkurat sitt fagområde. Finnes ikke en nødvendig regel er det ingen ting systemet kan gjøre.

3.3 Medisinske systemers akilleshæl: Håndtering av usikkerhet

Som det ble nevnt ovenfor var en av grunnene til at medisinske ekspertsystemer ikke ble tatt i bruk deres manglende evne til å håndtere usikkerhet (Heathfield 1999).

Usikkerhet i forhold til oppnådde konklusjoner er et spesiellt viktig problem for medisinske ekspertsystemer og da særlig i forhold til diagnostisering. Symptomer kan forekomme i kombinasjoner som gjør at ulike diagnoser kan være aktuelle. For eksempel kan en pasient ha enkelte symptomer som tyder på en diagnose men samtidig også ha andre symptomer som peker i en annen retning. Enda verre blir det hvis pasienten lider av flere sykdommer. Flere sykdommer i kombinasjon kan gi opphav til et helt annet symptommønster enn hvis sykdommene hadde opptrådt hver for seg. Pasienter som forteller om sine symptomer kan også legge til mye støy (Carroll 1987), det vil si informasjon som bare vanskeliggjør diagnostisering, ved at de kan legge for mye vekt på symptomer som de selv har lagt mest merke til, men som kanskje ikke er så viktige for en diagnose. Viktigere men ikke like fremtredende symptomer kan i verste fall være glemt av pasienten og derfor ikke nevnes. Dette er da informasjon man ikke har og som derfor heller ikke kan vurderes av et ekspertsystem.

3.3.1 Om håndtering av usikkerhet

Gitt at det er så vanskelig å gi helt sikre diagnoser, blir det viktig å kunne si hvor sikker man er på den diagnosen man har kommet fram til. For eksempel kan en lege være 80 prosent sikker på at en pasient har en gitt lidelse. Dette usikkerhetsanslaget er viktig fordi potensielt livsviktige beslutninger skal tas på bakgrunn av diagnosen, i alle fall i en medisinsk sammenheng. Hvilke behandlinger som skal velges, eventuelt om en behandling skal foretas i det hele tatt, avhenger jo av diagnosen. Et ekspertsystem som stiller medisinske diagnoser er derfor nødt til å gi informasjon om hvor sikker man kan være på dets konklusjoner.

En vanlig måte å representere usikkerhet i ekspertsystemer er gjennom såkalte sikkerhetsfaktorer. Som det ble sagt i kapittel to er sikkerhetsfaktorer tallverdier, typisk fra -1 til 1, som uttrykker systemets tiltro til en konklusjons sikkerhet. For eksempel kan et system konkludere med at en pasient har lungebetennelse med 0,75 sikkerhet.

Systemet beregner denne endelige sikkerhetsverdien på bakgrunn av sikkerhetsfaktorer knyttet til hver enkelt regel. Denne beregningen gjøres ved at de enkelte sikkerhetsfaktorene kombineres etter regler definert av systemutviklerne. Som et eksempel på en slik regel kan man ta en kombinasjonsregel fra Norvig (1992) sin forenklede Mycin variant. For å kombinere to regler som støtter samme konklusjon og der begge sikkerhetsfaktorene har en positiv verdi, plusser man sammen begge sikkerhetsfaktorene og legger til produktet av begge sikkerhetsfaktorene.

Det er viktig å merke seg at slike kombineringsregler ikke har noe å gjøre med sannsynlighetsregning og de matematisk utledede regnereglene som brukes der. Sikkerhetsfaktorer er altså ikke det samme som sannsynligheter. Derimot er disse reglene konstruert slik at de for det meste gir fornuftige resultater under kjøringer av systemet.

3.3.2 Problemer med tradisjonell usikkerhetshåndtering

De individuelle sikkerhetsfaktorene som hver enkelt regel er utstyrt med er bygget inn i kunnskapsbasen og kommer, som reglene selv, fra den eller de eksperter kunnskapsbasen er bygget på. Det vil si at ekspertene, under prosessen med å lage kunnskapsbasen, for hver regel må oppgi en tallverdi som gir en antyding om hvor sikre de er på en konklusjon gitt at de opplysningene som finnes i premissene foreligger. Hvis en pasient er slapp og huden til pasienten har en lett blåfarge (blå hudfarge er et tegn på at pasienten har for lite oksygen), hvor sikker er da eksperten på at denne pasienten har lungebetennelse? Hvor sikker er eksperten på andre konklusjoner man eventuelt også kan slutte fra disse symptomene?

For å gi et svar på dette gjør eksperten et usikkerhetsanslag. Dette anslaget vil være basert på ekspertens erfaring og er derfor subjektivt. Det vil si at en annen ekspert kunne kommet med et noe annet estimat.

Estimater av denne typen er typisk også intuitive. Det betyr at eksperten bare bruker sin egen kunnskap som grunnlag for estimatet. Vedkommende har ikke hjelp av statistiske data over lungebetennelsespasienter eller en formel for å beregne et usikkerhetsestimat. Eksperten gjør dette anslaget så og si på stående fot.

På denne måten oppnår man en verdi for hvor stor tiltro eksperten har til at en konklusjon er riktig dersom premissene er sanne. I vårt tilfelle kan eksperten for eksempel oppgi en sikkerhetsfaktor på 0,75 for diagnosen lungebetennelse dersom pasienten er slapp og har en blålig hudfarge. Siden en kunnskapsbase fort kan inneholde hundrevis av regler, må ekspertene som systemet modelleres etter også må ut med hundrevis av slike sikkerhetsanslag.

Å basere seg på ekspertenes sikkerhetsanslag er problematisk av to grunner. For det første bruker mennesker, også eksperter, en rekke kognitive strategier som i mange sammenhenger leder til systematiske feil, også ved anslag av sikkerhet (se for eksempel Hastie & Dawes 2001). For det andre og viktigere for oss i denne sammenhengen, er det faktum at mennesker har store vansker med å lære usikre sammenhenger gjennom erfaring (Brehmer 1980) og derfor heller ikke gjør gode sikkerhetsanslag.

Årsaken til at mennesker ikke lærer usikre sammenhenger gennom erfaring er å finne i menneskers tankesett. Mennesker liker ikke usikkre utfall og foretrekker å tro at hendelser er forutsigbare og kan forklares ut fra klare sammenhenger

mellom årsak og virkning. Denne deterministiske måten å tenke på synes å være svært grunnleggende for mennesker (Brehmer 1980).

Beklageligvis er ofte sammenhengen mellom årsak og virkning så kompleks og uoverskuelig at sammenhengen er bedre beskrevet som en probabilistisk sammenheng, det vil si at sammenhengen er utsatt for eller involverer tilfeldige variasjoner.

Men mennesker antar altså at forhold mellom variabler er deterministiske. Dette gjør at det er svært vanskelig for oss å tenke på en sammenheng som probabilistisk. Når man derfor ikke ser en sammenhengs probabilistiske natur, er det også umulig å behandle den tilsvarende. Da nytter det lite at det finnes prosedyrer og verktøy, som sannsynlighetsregning, for å håndtere probabilistiske sammenhenger. Istedenfor utvikler man egne deterministiske teorier om årsaksforhold mellom de aktuelle variablene gjennom de erfaringer man gjør. Disse teoriene kan ofte avvike fra de faktiske, probabilistiske sammenhengene mellom variabler og kriterier. Derfor er det også meget vanskelig å lære en usikker sammenheng av egen erfaring.

For utviklingen av en kunnskapsbase som skal ta høyde for usikkerhet innebærer dette at kunnskapsbasen risikerer å bli full av omtrentlige sikkerhetsanslag. Når disse anslagene kombineres i løpet av en programkjøring, etter regler laget for dette formålet, blir det svært uklart hvilken grad av sikkerhet de resulterende sikkerhetsfaktorene egentlig uttrykker.

3.3.3 Ekspertsystemer versus lineære regresjonsmodeller

Ekspertsystemtradisjonen har alltid sett på eksperter som idealet for menneskelig problemløsning. Derfor har man innenfor dette feltet aldri stilt spørsmålstegn ved hvor god ytelsen til eksperter faktisk er (Hammond 1996). Det som har vært av interesse er ekspertenes problemløsningsprosesser. Disse har man villet fange opp og bruke i ekspertsystemer.

Ekspertsystemtradisjonen er imidlertid ikke den eneste som har interessert seg for eksperter. Innenfor en gren av kognitiv psykologi, bedømmings- og beslutningspsykologien, har man også studert eksperter, men med et helt annet utgangspunkt enn ekspertsystemtradisjonen. I bedømmings- og beslutningspsykologien har man i større grad sammenlignet eksperters ytelser med statistiske regresjonsmodeller (Camerer & Johnson 1991). Startskuddet for denne psykologiske tradisjonen var Meehls bok «Clinical versus statistical prediction» fra 1954. Temaet for denne boken var at enkle regresjonsmodeller er i stand til å integrere informasjon bedre enn mennesker, også eksperter. Denne boken startet en forskningstradisjon der man satte eksperters prediksjoner opp mot prediksjoner fra regresjonsmodeller. Slike statistiske modeller er basert på registrerte sammenhenger mellom det kriteriet man vil predikere og relevante variabler. Senere begynnte man også å sammenligne eksperters prediksjoner med enklere uekte modeller, det vil si modeller der predikatorvariablene er vektet etter en uoptimal metode.

Hovedfunnene fra denne tradisjonen, langt over hundre studier av ekspert-prediksjoner og statistiske modeller, kan sammenfattes i tre punkter.

- Eksperter predikerer ikke så fryktelig bra.
- Enkle statistiske regresjonsmodeller er stort sett enten like gode eller bedre i sine prediksjoner enn ekspertene.

- Eksperter er, til tross for at de ikke klarer å bruke denne informasjonen like godt som en regresjonsmodell, gode til å finne frem til de variablene som er viktige for en prediksjon.

Et eksempel på et problem der en regresjonsmodell integrerer informasjon bedre enn eksperter er gitt av Dawes & Corrigan (1974) og omhandler vurdering av kandidater til opptak for videregående studier. Viktige variabler for denne vurderingen er resultatet av den såkalte Graduate Record Examination (GRE), en test som ofte blir avkrevd søkende studenter i engelskspråklige land, og studentens gjennomsnittskarakterer så langt i studieløpet. En konkret kandidat har en GRE på 750 og et gjennomsnitt på 3,3, mens en annen kandidat har en GRE på 680 men et gjennomsnitt på 3,7.

Hvilken av disse to kandidatene er best? For å finne ut dette med utgangspunkt i disse verdiene må man nødvendigvis kombinere denne informasjonen med hensyn til at den ene kandidaten har et høyere gjennomsnitt mens den andre kandidaten har bedre GRE resultater. Spørsmålet blir da hvordan man skal gjøre dette. De ulike verdiene lar seg ikke uten videre sammenligne. GRE resultater for interessante kandidater kan ligge et sted mellom 500 og 800, mens karaktergjennomsnittet kan ligge mellom 3,0 og 4,0.

Dawes & Corrigan (1974) gjorde en regresjonsanalyse basert på gjennomsnittskarakter, GRE resultater, et mål på kvaliteten til skolene kandidatene kom fra og et mål på på kandidatenes suksess ved universitetet noen år senere. Denne ekte modellen slo en utvalgskomité til tross for at denne komitéen hadde tilgang til mer informasjon enn modellen. De hadde et anbefalingsbrev fra studentens opprinnelige skole og hadde også intervjuet kandidatene.

Dawes & Corrigan (1974) fant også at både GPA og målet for de opprinnelige skolenes kvalitet hver for seg korrelerte høyere med målet på kandidatenes senere suksess enn utvalgskommiteens prediksjoner. Man ville altså slått utvalgskommiteen selv ved bare å basere seg på en av disse variablene.

Eksempelet ovenfor er bare et av mange som illustrerer at lineære regresjonsmodeller klarer å levere mer nøyaktige prediksjoner enn eksperter. Grunnen til dette er at selv enkle statistiske modeller er bedre til å integrere den relevante informasjonen enn eksperter. Eksperter på sin side har masse informasjon men klarer ikke bruke denne informasjonen like godt som en statistisk modell fordi de ofte baserer sin problemløsning og beslutningstaking på såkalte konfigurale regler (Camerer & Johnson 1991).

En konfigural regel er en regel der en variabels viktighet avhenger av andre variabler. Eksempelvis kan man ha en regel som sier at man bare ansetter en stipendiat dersom vedkommende har et glødende anbefalningsbrev, glimrende karakterer og et interessant forskningsprosjekt. I følge Camerer & Johnson (1991) bruker eksperter slike regler fordi de er lettvinte eller mindre anstrengende. Regelen ovenfor sier at man bare ansetter folk som er gode i alle de tre kategoriene. Ved å gjøre dette slipper man å veie de ulike variablene opp mot hverandre. Dette er langt mindre anstrengende enn å måtte kombinere informasjonen til en kandidat som har et interessant forskningsprosjekt men som bare har middelmådige karakterer.

Disse konfigurale reglene er imidlertid ofte unøyaktige (Camerer & Johnson 1991). Det vil si at de i liten grad korrelerer med kriteriet man forsøker å komme frem til. Alle premissene ekspertene bruker i sine kompliserte konfigurale regler er med andre ord ikke alltid like gode prediktorer. Dette skyldes at disse reglene

ofte er basert på et lite utvalg og derfor raskt overgeneraliseres og brukes på tilfeller de ikke passer for. Konfigurale regler kan også være basert på feilaktige forestillinger om forhold mellom variabler.

Eksperters bruk av lettvinte men unøyaktige konfigurale regler gjør altså at lineære regresjonsmodeller utkonkurrerer dem i mange tilfeller. I tilegg til dette kan eksperter, som alle andre, ha dårlige dager, være trøtte, sultne eller opptatt av personlige problemer. Slike tilstander kan gi negativ utslag på ekspertens ytelse. En lineær modell vil derimot alltid gi det samme svar gitt den samme input (se for eksempel Dawes, Faust & Meehl 1989).

I tillegg til at lineære regresjonsmodeller ofte slår eksperters ytelse er de også både enkle og billige å lage. Det tar ikke stort mer enn en dag å lage en regresjonsmodell av en ekspert (Hammond 1996). Derimot er ekspertsystemer både tidkrevende og dyre å utvikle. I følge Carroll (1987) kreves det investeringer i millionklassen før man i det hele tatt får systemet til teststadiet. Når det gjelder utviklingstiden var fem år ansett som et minimum på 1980-tallet (Davis 1982, Carroll 1987). I tillegg til å være både dyre og tidkrevende i produksjon er ekspertsystemer i *beste fall* bare i stand til å oppnå en ytelse som tilsvarer ytelsen til eksperten det er bygget på (Carroll 1987). De klarer altså ikke å overgå den orginale eksperten.

Bedømmings- og beslutningspsykologiens studier av eksperters ytelser i forhold til lineære modeller er av betydning for ekspertsystemfeltet på to måter. For det første gir lineære modeller eksakte sikkerhetsestimater siden de er bygget på faktiske, registrerte sammenhenger mellom variabler. For det andre gjør de det, ved å være så billige og enkle å lage, vanskelig å rettferdiggjøre utviklingen av store og komplekse ekspertsystemer som allikevel yter suboptimalt i forhold til både eksperten det er bygget på og en lineær modell basert på den samme eksperten.

Men disse funnene fra bedømmings- og beslutningspsykologien har hatt liten innvirkning på ekspertsystemtradisjonen. Dette skyldes at dette er to ulike fagfelt som i stor grad ikke kjenner til hverandres arbeider og som skjelden eller aldrig snakker sammen (Hammond 1996).

3.4 Sammenfatning

I dette kapittelet har jeg sett på problemer ved ekspertsystemer på 1980-tallet. Noen av disse problemene gjaldt alle ekspertsystemer. Systemene var vanskelig tilgjengelige for brukerne. De var også tunge å oppdatere samtidig som regelmessige oppdateringer var avgjørende for at systemene skulle kunne fungere slik de var tenkt. Ekspertsystemene på denne tiden var også preget av plutselige fall i ytelse dersom brukere forsøkte å løse problemer som ikke var påtenkt med regler i kunnskapsbasen.

Et annet generelt problem for alle ekspertsystemer var den tungvinte og tidkrevende prosessen med å lage kunnskapsbaser. Dette ble vanskeliggjort av eksperters manglende evne til å snakke om sin problemløsning og systemutviklernes manglende evne til å forstå ekspertens fagsjargong.

Et problem for ekspertsytstemtradisjonen var også funn fra bedømmings- og beslutningspsykologien som viser av eksperter ikke presterer veldig bra. Eksperter er flinke til å finne de viktige variablene man trenger for å løse et problem, men de overgås som regel av enkle statistiske modeller når det gjelder å kombinere informasjonen fra disse variablene. Dette gjør det vanskeligere å

rettferdiggjøre utviklingen av ekspertsystemer siden lineære regresjonsmodeller både er langt billigere og mye mindre tidkrevende å lage.

Medisinske ekspertsystemer var i en særklasse som spesielt problematiske på 1980-tallet. Mens enkle ekspertsystemer tatt i bruk av lønnsomhetsorienterte bedrifter gjorde suksess på denne tiden, spilte mer avanserte medisinske systemer fallitt. Dette skyldes at de medisinske systemene ofte ble laget som rene eksperimenter der systemutviklerne var mindre interessert i om systemene ble tatt i bruk eller ikke. I tillegg var medisinske ekspertsystemer utsatt for en svært skeptisk brukergruppe som fryktet å miste prestisje til intelligente systemer.

Et stort problem for de medisinske systemene var imidlertid problemet rundt usikkerhetshåndtering. Det ble påpekt at det er problematisk å få eksperter til å oppgi sikkerhetsanslag for en konklusjons sannhet fordi eksperter ikke lærer seg den statistiske sammenhengen mellom variabler i problemområdet gjennom erfaring. Siden ekspertsystemers usikkerhetshåndtering baserer seg på eksperters usikkerhetsanslag er denne usikkerhetshåndteringen mangelfull i utgangspunktet. En grundigere tilnærming ble funnet hos bedømmings- og beslutningspsykologien. Her har man en lang tradisjon med å lage lineære regresjonsmodeller for vurderinger. Disse modellene er basert på faktiske forhold mellom kriteriet man vil forutsi og de data som faktisk forutsier dette kriteriet. Dette gjør at sannsynlighetsanslag fra slike aktuariske modeller er bedre forankret i virkeligheten enn eksperters subjektive og intuitive anslag.

I neste kapittel skal jeg så se på hva fremveksten av Internett har gjort for å løse disse problemene.

4 Internett og ekspertsystemer

I løpet av de siste årene har det dukket opp en rekke nettbaserte ekspertsystemer (se for eksempel Grove 2000, Duan et al. 2005, Huang & Chen 2007). Tydeligvis gjør internettets muligheter ekspertsystemkonseptet mer interessant igjen. Skyldes denne økte interessen for ekspertsystemer at noen av de gamle problemene fra 1980-tallet løses?

Dette kapittelet er delt i to deler. I første del gir jeg to eksempler på internettbaserte ekspertsystemer.

I andre del av kapittelet tar jeg for meg spørsmålet om Internett faktisk gir ekspertsystemene nye muligheter eller om de gamle problemene består på tross av Internettrevolusjonen.

4.1 Internettbaserte ekspertsystemer: To eksempler

Vi skal her se på to representer for de nye ekspertsystemene. The Reptile Identification Helper er et system for å identifisere ulike reptilarter i Pennsylvania. Dette systemet fungerer som et tradisjonelt ekspertsystem bortsett fra at det opererer fra en nettbasert plattform. Det andre systemet vi skal se på er Fish-expert, et ekspertsystem for diagnose av fiskesykdommer. Det særegne med Fish-expert er, som vi skal se, innovativ bruk av Internett i et forsøk på å gjøre ekspertsystemet mer robust.

4.1.1 Reptile Identification Helper

The Reptile Identification Helper (Grove 2000) er et internettbasert ekspertsystem for å identifisere ulike slag reptiler og finnes tilgjengelig for alle under nettadressen http://grove.cs.jmu.edu/parih/index.jsp. Dette systemet ble utviklet i forhold til telling av ulike reptilslag i delstaten Pennsylvania, U.S.A. Systemets ekspertise er kunnskap om hvordan forskjellige reptiler i Pennsylvania ser ut. Denne ekspertisen hjelper så brukerne av systemet med å artsbestemme reptiler de kommer over. Brukerne registrerer så de reptilene de har sett i tellingen.

Dette systemet er installert på en tjenermaskin og en bruker kommer i kontakt med systemet ved å oppgi tjenerens adresse til en nettleser på sin egen lokale datamaskin. Dette forutsetter naturligvis at brukeren har en internettforbindelse.

Når brukeren har koblet seg opp mot tjeneren kan vedkommende begynne selve ekspertsystemsesjonen. For å identifisere et dyr stiller systemet maksimum ti spørsmål rundt forskjellige artsspesifikke kjennetegn. Siden man vil at så mange som mulig skal kunne være med å registrere reptilarter, er hvert spørsmål ledsaget av bilder av de ulike kjennetegnene. Brukeren trenger derfor ikke å være kjent med ordene som brukes for å benevne disse kjennetegnene. Enkelte ord er også utstyrt med lenker som leder til forklaringer av det aktuelle ordet. Bildene som vises frem kan også forstørres ved et museklikk.

Etter at brukeren slik har svart på en del spørsmål om hvordan det aktuelle dyret ser ut, kommer systemet frem til en eller flere mulige konklusjoner. Systemet presenterer også et sikkerhetsestimat som sier hvor sikker den er på at brukeren mener dette dyret. Deretter kan brukeren registrere det identifiserte reptilslaget i tellingen.

4.1.2 Et ekspertsystem for diagnose av fiskesykdommer

Duan et al.'s (2002) fiskeekspertsystem, kalt Fish-expert, er et eksempel på et industrielt mer nyttig system. Problemområdet systemet omhandler er sykdommer i fiskebestander og er tiltenkt brukt som diagnostiseringsverktøy for fiskeoppdrettere.

Motivasjonen bak Fish-expert er situasjonen i den kinesiske oppdrettsnæringen. På grunn av storstilet og tildels ukontrollert utbygning av oppdrettsanlegg og på grunn av et allment lavt kunnskapsnivå hos oppdretterne, er fiskesykdommer blitt et alvorlig problem i Kina (Duan et al. 2002, Duan, Fu & Li 2003). Ofte dør hele bassenger med fisk som følge av sykdommer. Dette har naturligvis store økonomiske konsekvenser for oppdretteren.

For å forhindre store tap kreves det ofte at en behandling påbegynnes umiddelbart etter at en sykdom er påvist. Derfor må en diagnose foreligge så raskt som overhodet mulig. Siden diagnostisering av sykdommer hos fisk er en komplisert affære hvor mange ulike forhold må tas i betraktning, er oppdrettere sjelden i stand til å diagnostisere selv. Derfor er de avhengige av at veterinærer kommer og foretar en diagnose på stedet. To forhold gjør denne avhengigheten problematisk. For det første finnes det for få veterinærer til at alle oppdrettere kan sikres eksperthjelp dersom sykdommer oppdages. For det andre ligger de fleste oppdrettsanlegg utilgjengelig til ute på landsbygda, langt unna de statlige veterinærsentrene.

Fish-expert har derfor som mål å hjelpe oppdrettere til selv å stille diagnoser på sine fiskebestander slik at behandling kan starte så raskt som mulig.

Fish-expert ligger, i likhet med reptilidentifikasjonssystemet, på en sentral tjener som brukere får tilgang til gjennom nettlesere installert på sine egne maskiner. Systemet viser frem tekst og bilder som beskriver ulike symptomer. Brukeren kan så velge de tekster og bilder som beskriver symptomene vedkommendes fisk synes å ha. Systemet prøver på basis av dette å komme frem til hvilken sykdom som er den aktuelle.

Fish-expert klarer imidlertid ikke alltid å løse alle problemer. I følge Duan et al. (2003) diagnostiserer Fish-expert 80 prosent av sykdommene. For de resterende tilfellene klarer systemet ikke å levere en diagnose da det ikke har noen kunnskap om dem. For disse tilfellene har Duan et al laget et system hvor brukeren over Internett kan komme i kontakt med en veterinær som kan bistå brukeren med diagnostisering. Dette systemet, kalt T-vet, er integrert i ekspertsystemet og tillater i beste fall at brukeren kan kommunisere i sanntid med en veterinær.

Sanntidskommunikasjon via T-vet forutsetter at den finnes minst en ekspert som er logget inn i systemet. Innloggede oppdrettere kan så velge hvilken ekspert de ønsker å kontakte. Selve samhandlingen mellom oppdretter og veterinær foregår i et såkalt virtuelt diagnoserom der oppdretteren kan formidle symptomer via bilder eller tekst og der veterinæren kan svare på oppdretterens spørsmål.

Dersom det ikke befinner seg noen eksperter innlogget kan oppdrettere i nød benytte seg av det Duan et al. (2003) kaller for T-vets asynkrone diagnostiseringssystem. Dette fungerer slik at oppdretteren fyller ut et elektronisk symptomskjema med de aktuelle symptomene og derpå logger seg ut av systemet. Når en veterinær logger seg inn i det asynkrone diagnostiseringssystemet kan vedkommende gå gjennom de forespørslene som foreligger å besvare dem via

e-post. Så vel oppdretterens forespørsel og veterinærens diagnose blir lagret i en database. Denne informasjonen kan senere brukes til å forbedre Fish-Experts kunnskapsbase. Det asynkrone diagnostiseringssystemet har også den fordelen at det ikke krever en internettforbindelse med høy hastighet eller digitalt videoutstyr, noe som gjerne kreves dersom man vil overføre digitale bilder.

4.2 Ekspertsystemenes problemer og internetts løsninger

4.2.1 Oppdateringsproblemet

Som det ble påpekt i forrige kapittel var oppdatering av kunnskapsbasen et problem på 1980-tallet. Fordi ekspertsystemer er avhengige av kunnskap er det viktig å kunne forandre kunnskapsbasen også etter at hovedsystemet er ferdig utviklet. Det kan jo lett skje at man oppdager feil eller mangler i kunnskapsbasen og må rette opp disse. Dersom man oppnår ny viten som er viktig for et ekspertsystems fagområde er det også viktig at kunnskapsbasen får kjennskap til denne kunnskapen ved at nye regler legges til.

Denne oppdateringen er noe som Internett forenkler enormt i forhold til situasjonen på 1980-tallet. Som regel vil ekspertsystemet i all hovedsak kjøres på en tjenermaskin som eventuelle brukere kobler seg opp mot (Grove 2000). Oppdateringer kan så gjøres direkte på tjeneren. Man slipper å oppdatere hundrevis av individuelle systemer via postforsendelser av oppdateringsdisketter eller å lage brukervennlige oppdateringer som brukerne laster ned og installerer selv. Brukerne på sin side vil merke lite eller ingen ting til oppdateringsaktiviteten siden de selv ikke trenger å gjøre noe på sine lokale maskiner (Duan et al. 2005).

4.2.2 Tilgjengelighetsproblemet

Ekspertsystemers tilgjengelighet ble sett på som begrenset på 1980-tallet. Dette er også et problem som Internet kan bidra til å forbedre betraktelig fordi Internet i seg selv er lett tilgjengelig (Duan et al. 2005, Grove 2000). Gitt den tilsvarende infrastrukturen har man i teorien tilgang til Internet over hele verden. Mulighet for trådløse nett mange steder gjør at nettbaserte ekspertsystemer kan tenkes å kjøres fra bærbare datamaskiner under feltbetingelser. Grove (2000) nevner til og med NASAs planer for et nettverk utover i solsystemet. Slik er ekspertsystemenes kunnskap tilgjengelig på det sted og til den tid det trengs. Brukere av nettbaserte ekspertsystemer er slik ikke avhengige av å befinne seg på den maskinen der systemet er installert.

Ta for eksempel den visiterende allmennlegen vi nevnte i forrige kapittel. Denne legen trenger ideelt sett ikke å bevege seg vekk fra pasientens seng for å innhente spesialisthjelp. Dersom legen kan kommunisere med et ekspertsystem via en bærbar datamaskin kan vedkommende uten videre få tilgang til den ekspertisen som trengs.

4.2.3 Problemet med kunnskapsakvisisjonen.

Et problem i utviklingen av ekspertsystemer er kunnskapsbasens tilegnelse av kunnskap. Som beskrevet før er denne prosessen den største flaskehalsen i arbeidet med å utvikle et ekspertsystem og er hovedgrunnen til at utviklingstiden til ekspertsystemer er så lang.

Denne omstendelige prosessen hjemsøker imidlertid ikke bare systemutviklerne under utviklingen av systemet. Som nevnt i forrige kapittel kreves det at kunnskapsbasen hele tiden holdes ved like. En metode som kunne forenkle systemets tilegnelse av kunnskap skulle derfor ikke bare lette utviklingen av systemet men også gjøre oppdateringer av kunnskapsbasen enklere.

Enkelte mener at Internet kan bidra til å lette denne prosessen. I et forsøk på å stake ut kursen videre for ekspertsystemutviklingen peker Buchanan et al. (2006) nettopp på muligheten for å kunne hente inn informasjon fra Internett. Disse forfatterne har høye forventninger til at dette skal la seg muliggjøre ved hjelp av såkalt semantisk web. Semantisk Web er en videreutvikling av World Wide Web der informasjon som per i dag kun kan brukes av mennesker også kan tolkes av maskiner. Dette innebærer at informasjon som ligger på Internett, enten det dreier seg om bilder eller tekst, må kodes med en slags meningskode slik at denne informasjonen blir meningsbærende også for en maskinell leser. Hvis dette blir gjort i stor utstrekning kan man tenke seg at ekspertsystemer ville kunne lete opp relevant informasjon på Internett selv. Slik kan man forestille seg kunnskapsbaser som selvstendig finner den nyeste informasjonen og integrerer denne i nye regler. Eller ekspertsystemer som selvstendig leter etter mer bakgrunnskunnskap for å supplementere sin egen base. Denne utviklingen er imidlertid ikke kommet så langt at ekspertsystemer kan hente ned informasjon uten menneskelig hjelp (Buchanan et al. 2006).

Imidlertid har internett blitt brukt for å hjelpe til i innehentingen av kunnskap til kunnskapsbasen. Under utviklingen av ekspertsystemet for diagnostisering av fiskesykdommer brukte man et nettbasert vektøy for å hjelpe til med å opprette kunnskapsbasen (Duan et al. 2005). Dette verktøyet hadde et brukergrensesnitt som ekspertene kunne bruke for å oppgi data om sykdommers symptomer, preventive tiltak og behandlingsmetoder til kunnskapsbasen. Ekspertene kunne også bruke dette grensesnittet for både å legge til og forandre informasjon eller regler i kunnskapsbasen (Duan et al. 2005). Et lignende verktøy rapporteres det om i Huang & Chen (2007). Her kunne ekspertene legge til symptomer, sikkerhetsfaktorer, mulige behandlinger og all annen relevant informasjon som trengtes for å lage regler. Ekspertene kunne også her oppdatere informasjon eller regler som allerede fantes i systemet.

Ekspertene hadde i begge disse tilfellene tilgang til all informasjon i systemene og de kunne også endre denne. Dette betyr at det finnes en mulighet, enten den ble benyttet i disse to systemene eller ikke, for at eksperter kan sitte ved sine egne maskiner på sine egne kontorer og derfra bidra med kunnskap til en kunnskapsbase. Slike verktøy gir også muligheter for at ekspertene kan overprøve systemet og endre på systemets regler helt til de gir et tilfredsstillende resultat. Denne muligheten for å kunne hjelpe til med systemet fra avstand gjør det muligens lettere å bruke eksperter som holder til geografisk langt vekk fra systemutviklerne. Det burde også gjøre det enklere å bruke flere eksperter i den grad et slikt nettbasert grensesnitt minsker nødvendigheten for å møte ekspertene fysisk.

Hos både Huang & Chen (2007) og Duan et al. (2005) kom imidlertid bruken av disse nettbaserte grensesnittene ved siden av vanlige intervjuer med eksperter. For eksempel brukte Huang & Chen (2007) sitt grensesnitt først etter at man hadde opprettet de mest høynivå konseptene og begrepene innenfor kunnskapsbasen med den mer tradisjonelle intervjutilnærmingen. Det nettbaserte grensesnittet ble deretter brukt for å finne ut av detaljspørsmål

som gjensto. Videre trengtes det i begge tilfeller en systemutvikler som integrerte ekspertens informasjon som ekspertene oppgav gjennom grensesnittet i kunnskapsbasen. Systemutvikleren måtte kontrollere for at ikke redundans, sirkularitet eller ufullstendige regler snek seg inn i kunnskapsbasen.

Både Huang & Chen (2007) og Duan et al. (2005) hevder at denne nettbaserte måten å samle inn data på, både hjalp ekspertene til å tydeliggjøre sine resonneringsprosesser og hjalp systemutviklerne å representere kunnskapsområdet på en god måte. Men ingen av dem opplyser om denne nettbaserte kunnskapsakvisisjonen faktisk har forenklet prosessen eller ikke. Sparte man for eksempel mer tid på å bruke nettbasert kunnskapsakvisisjon? Eller ble den resulterende kunnskapsbasen bedre enn den ville blitt med kun den tradisjonelle tilnærmingen?

Det virker allikevel trolig at slik kunnskapsakvisisjon over Internett kan spare systemutviklere for en del anstrengelser, spesielt når det gjelder å holde kunnskapsbasen oppdatert. Eksperter kan oppdatere systemet fra hvor enn de måtte befinne seg og systemutvikleren kan kontrollere denne oppdateringen uten å måtte oppsøke eksperten.

4.2.4 Lite robuste systemer

Et videre problem med 1980-tallets ekspertsystemer var disse systemenes manglende robusthet. Dette innebar at ekspertsystemer falt voldsomt i ytelse dersom de måtte løse problemer som det ikke fantes regler for i deres kunnskapsbaser. Dette ytelsesfallet kommer av at ekspertsystemer, i motsetning til deres menneskelige motstykker, ikke har generell bakgrunnskunnskap som man kan slutte konklusjoner fra når spesialistkunnskapen ikke strekker til (McCarthy 1984).

Buchanan et al. (2006) mener at generell kunnskap ikke er nok for å gi ytelser på ekspertnivå innenfor spesialiserte områder, men at det likefullt er nødvendig at et ekspertsystem har slik kunnskap. Denne bakgrunnskunnskapen er av en slik art som ethvert menneske vil være i besittelse av. For eksempel vet alle mennesker at et rom har et gulv. Et ekspertsystem kan derimot bare vite dette dersom det finnes en tilsvarende regel i en kunnskapsbase systemet har tilgang til. Tradisjonelle kunnskapsbaser ville fort bli uhorvelig store om man skulle bygge inn bakgrunnskunnskap med dette detaljnivået.

Kan Internett bidra til å gi ekspertsystemer slik bakgrunnskunnskap? Grove (2000) mener dette er en mulighet. Han tenker seg at ulike ekspertsystemer kan samarbeide over Internett og slik til en viss grad dele sine kunnskapsbaser. Hvis et system mangler regler for et problem kan det spørre et annet system om hjelp eller eventuelt henvise brukeren til dette andre systemet. Dette gjøres også enklere av ekspertsystemskallarkitekturen der kunnskapsbasen er en separat modul i forhold til slutningsmekanismen. Man kan tenke seg at et ekspertsystemskall slik kan bytte over til en annen kunnskapsbase dersom den basen det nå bruker ikke strekker til. Eller man kan opprette kunnskapsbaser som mange ekspertsystemskall deler. Dette forutsetter naturligvis at ulike kunnskapsbaser er laget på samme form. Et eksempel, gitt av Buchanan et al. (2006), er at mange ekspertsystemer som omhandler ulike trusler mot kulturplanter trenger litt kunnskap om insekter. Disse systemene kunne da delt sine kunnskaper om insekter via Internett.

En annen tilnærming for å gi ekspersystemer bakgrunnskunnskap nevnes

av Buchanan et al. (2006). Disse forfatterne setter store forhåpninger til forsøk på å lage store databaser av allmennkunnskap. Det henvises til CYC (http://www.cyc.com/), en gigantisk kunnskapsbase av bakgrunnskunnskap. Denne kunnskapsbasen ble påbegynt i 1984 og inneholder nå omtrent to hundre tusen begreper. Til hvert begrep finnes det flere regler som omhandler eller bruker dette begrepet. Disse begrepene omhandler helt grunnleggende menneskelig kunnskap. Tanken er at ekspertsystmer kan bruke slike store allmenne kunnskapsbaser til å lære seg bakgrunnskunnskap som er relevant i forhold til deres problemløsning ved hjelp av maskinlæringsteknikker. Buchanan et al. (2006) innrømmer imidlertid at det fortsatt er uklart om ekspertsystemers bruk av kunnskapsbaser som CYC faktisk vil lede til smartere, det vil si mer robuste, systemer. Hvis bruk av allmenne kunnskapsbaser skulle vise seg anvendelig i fremtiden gjør naturligvis Internett kommunikasjon mellom enkelte ekspertsystemer og almenne kunnskapsbaser mulig.

Kommunikasjon mellom ekspertsystemer og eksterne kunnskapsbaser forutsetter imidlertid felles prosedyrer for kommunikasjon. Grove (2000) peker på at det finnes protokoller for utveksling av informasjon og tjenester mellom programmer over Internett. Disse må imidlertid tilpasses ekspertsystemers bruk. Eventuelt kan man lage egne protokoller basert på XML (Grove 2000). XML er et generelt oppmerkingsspråk hvis hovedhensikt er å lette deling av data mellom ulike systemer, særlig over Internett. XML kan brukes til å lage oppmerkingsstrukturer for spesielle anvendelsesområder. For eksempel kan man si om et element i en fil at dette er en regel. Innenfor denne regelen kan man så ha merket opp hvilke elementer som er premisser og hvilket element som er enkonklusjon. Programmet som får tilsendt denne filen kan så lese innholdet og tolke dette tilsvarende såfremt programmet har tilgang til den rette definisjonen. Ekspertsystemer kan slik få oversendt hele kunnskapsbaser på dette formatet og bruke disse til å løse sine problemer. Et problem med dette er at responstiden til et ekspertsystem går ned dersom det må kommunisere med et annet system for å supplementere sin egen kunnskapsbase.

Det mest konkrete eksempelet på bruk av Internett for å styrke et systems robusthet er vist av Duan et al. (2005). Disse forfatterne innrømmer at systemet ikke alltid klarte å håndtere kompliserte problemer på grunn av begrensninger i kunnskapsbasen. Dette løste systemet, som nevnt ovenfor, ved at brukeren, gjennom ekspertsystemet, kunne ta kontakt med en ekspert og fremlegge det aktuelle problemet for denne dersom systemet ikke klarte å komme frem til noen fornuftig løsning.

En slik mulighet til å konsultere en ekspert forutsetter naturligvis at man har eksperter til rådighet som kan gjøre denne tjenesten. Dette gjør at denne løsningen ikke er fullgod. For i mange sammenhenger kan det nok være vanskelig å organisere en slik «eksperttelefonvakt» som står klar til å hjelpe brukere. Men der en slik beredskap er mulig skulle ekspertsystemer som Fish-expert likevel avlaste ekspertene samtidig som ekspertenes ekspertise blir gjort lettere tilgjengelig for allmennheten.

4.2.5 Problemet med resonnering under usikkerhet.

Et stort problem for spesielt medisinske ekspertsystemer på 1980-tallet var deres mangelfulle usikkerhetshåndtering. Grunnen til dette er i hovedsak at eksperter sjelden klarer å lære problabilistiske sammenhenger gjennom erfaring

og at sikkerhetsfaktorene tilknyttet kunnskapsbasens regler derfor kan bli vage. Dette forsterkes av at ekspertsystemers kombineringsregler ikke er basert på noen objektiv metode for kombinering.

Denne problemstillingen gjør naturligvis internett i seg selv ikke noe med. Eksperter er fortsatt dårlige til anslå usikkerheten av konklusjoner fordi de fortsatt ikke lærer den nødvendige informasjonen fra sin erfaring, Internett eller ei. Et av hovedproblemene for ekspertsystemtradisjonen forblir derfor uløst og paradokset i forhold til de enkle statistiske modellene består. Det vil si at disse modellene som oftest er mer nøyaktige i sine anslag enn kompliserte og dyre ekspertsystemer.

Internett gir imidlertid muligheter for å kombinere det beste fra begge tradisjoner. Ekspertsystemene bidrar med forklaringsmuligheter og informasjonsinnhenting mens lineære regresjonsmodeller bidrar med integrering av usikkre data og nøyaktige anslag. Hvis ekspertsystemer for eksempel kan hente statistisk informasjon fra Internett selv ved hjelp av semantisk web eller lignende, kan ekspertsystemet så bygge statistiske modeller på bakgrunn av denne informasjonen og bruke disse anslagene (Buchanan et al, 2006). En annen mulighet er at ekspertsystemer kan isolere de problemene der kombinering av informasjon kreves å bruke egne statistiske moduler for å løse disse.

4.3 Sammenfatning

I dette kapittelet har vi sett på to representanter for de nye, nettbaserte ekspertsystemene, nemlig Reptile Identification Helper og Fish-expert. Reptile Identification Helper hjelper brukeren å fastslå artstilhørigheten til reptiler mens Fish-expert diagnostiserer sykdommer i fiskebestander. Vi så at Fish-expert inneholder en mulighet for kontakt med en menneskelig ekspert via Internett dersom ekspertsystemet ikke klarer å diagnostisere en sykdom.

Vi så videre på hvilke av problemene fra 1980-tallet som Internett bidrar til å løse.

Det ble påpekt at Internett gjør ekspertsystemer langt enklere å holde oppdatert enn det som var tilfellet før. Oppdateringer kan gjøres direkte på tjenermaskinen uten at brukerne trenger å innvolveres.

Internett har også bidratt til å gjøre ekspertsystemer mer tilgjengelige enn de var på 1980-tallet. Gjennom Internett kan nettbaserte ekspertsystemer brukes fra alle steder hvor man har nettilgang.

Når det gjaldt kunnskapsakvisisjonen så vi at det finnes forhåpninger til at semantisk Web kan muliggjøre at ekespertsystemer selv finner informasjon på Internett. Dette er imidlertid ikke mulig enda. Det mest konkrete vi så når det gjelder å bruke Internett for å lette kunnskapsakvisisjon, var gjennom bruk av nettbaserte brukergrensesnitt som gjorde det mulig for eksperter å bidre med informasjon til kunnskapsbasen.

Vi så at enkelte har håp om at store kunnskapsbaser med bakgrunnskunnskap skal kunne bedre ekspertstsystemers robusthet. En annen lansert mulighet er å dele kunnskap mellom ekspertsystemer via Internett. Styrking av robusthet, ved hjelp av Internett, i en faktisk implementasjon så vi i Fish-expert hvor menneskelige eksperter til nøds kan hjelpe brukerne.

Problemet med ressonnering under usikkerhet løser internett ikke siden dette er et problem som dypest sett skyldes måten mennesker tenker på.

5 Sammenfatning og konklusjon

5.1 Sammenfatning

I denne oppgaven har vi først kastet et raskt blikk på ekspertsystemfeltet og dets opprinnelse innenfor kunstig intelligensforskningen på 1950-tallet. Interessen for ekspertsystemer som spesifikke heller enn generelle problemløsere, ble større etter at generelle problemløsningsprogrammer ikke hadde klart seg særlig bra mot kompliserte problemer. Vi så at ekspertsystemfeltet etter hvert ikke manglet store visjoner og at suksessen til enkelte tidlige ekspertsystemer bidro til å styrke engasjementet rundt systemer av denne typen.

Vi gikk videre gjennom hvordan et typisk ekspertsystem er bygget opp. Et minimalt ekspertsystem består av en slutningsalgoritme og en kunnskapsbase.

Kunnskapsbasen inneholder kunnskap om et problemområde organisert i regler og bygges opp ved hjelp av en eller flere menneskelige eksperter i samarbeid med en eller flere systemutviklere. Vi så imidlertid at det ikke er helt uproblematisk å få tak i ekspertens kunnskap da deler av denne kunnskapen er ubevisst og fordi eksperten er vant til å uttrykke seg i et fagspråk som systemutvikleren sjelden har kjennskap til.

Den andre helt nødvendige komponenten i et ekspertsystem er slutningsmekanismen. Slutningsmekanismen er den delen av ekspertsystemet som forsøker å løse problemer ved hjelp av kunnskapen fra kunnskapsbasen. Mange slutningsmekanismer fungerer ved baklengs kjeding der hvert premiss i en regel behandles som et delmål på veien til å løse hovedproblemet. Dersom ekspertsystemet må kunne resonere under usikkerhet er det slutningsmekanismens oppgave å sørge for dette ved å beregne en endelig sikkerhetsfaktor for konklusjonen. Det er også viktig at slutningsmekanismen er gjennomsiktig slik at brukerne kan forstå hvordan den går frem for å løse et problem.

Siden alle ekspertsystemer trenger slutningsmekanismer fant man på å lage generelle slutningsmekanismer som kan bruke kunnskapsbaser fra ulike fagområder. En slik uavhengig slutningsmekanisme er hovedelementet i det som kalles for et ekspertsystemskall.

Vi har videre sett at interessen for ekspertsystemene dalte på slutten av 1980-tallet til tross for de store vyene og det store engasjementet som preget feltet i startfasen. Dette skyldtes at ekspertsystemene på denne tiden slet med flere problemer:

- Det var veldig tidkrevende å lage systemenes kunnskapsbaser. Dette bidro igjen til svært lange utviklingstider slik som vi så med Caduceus.
- Ekspertsystemenes tilgjengelighet var dårlig. Systemene var fysisk bundet til noen få datamaskiner og man var nødt til å oppsøke en av disse for å bruke dem.
- Systemene var vanskelige å oppdatere i fall man skulle oppdage feil i de eksisterende reglene eller at man trengte å innlemme nyoppdaget viten i kunnskapsbasen. Som vi har sett måtte disse oppdateringene gjennomføres ved postforsendelser av disketter til brukerne.
- Systemene var ofte lite robuste og fallt drastisk i ytelse dersom de ble forsøkt brukt på uforutsette måter.

I tillegg til disse allmenne problemene så vi at medisinske ekspertsystemer var spesielt problematiske:

- Medisinere var skeptiske til å ta systemene i bruk.
- Det medisinske fagområdet ble ofte brukt som et interessant problemfelt for uttesting av nye teknikker heller enn at systemutviklerne var opptatt av å utvikle levedyktige systemer.
- Systemene maktet ikke å håndtere usikkerhet på en god måte. Dette skyltes at sikkerhetsfaktorene som er knyttet til hver regel blir gitt av ekspertene som kunnskapsbasen bygges på. For å kunne komme med et slikt estimat støtter ekspertene seg på sine erfaringer innenfor det aktuelle problemområdet. Funn fra bedømmings- og beslutningspsykologien viser imidlertid at det er svært vanskelig for mennesker å lære probabilistiske sammenhenger av erfaring. Derfor kan ekspertenes sikkerhetsanslag ofte være vage i utgangspunktet. Når så ekspertsystemet kombinerer disse anslagene etter regler som ikke er laget på noe annet grunnlag enn at de gir fornuftige verdier ved programkjøring, er det svært uklart hva den endelige sikkerhetsfaktoren egentlig betyr.

Etter fremveksten av Internett har det, til tross for problemene nevnt ovenfor, dukket opp flere internettbaserte ekspertsystemer. Vi så på to eksempler på denne nye generasjonen ekspertsystemer: The Reptile Identification Helper og Fish-expert.

De nye internettbaserte ekspertsystemene var motivasjonen for å se om og hvordan Internett kan brukes til å løse de gamle problemene som ekspertsystemtradisjonen har møtt.

Som vi så gjør Internett oppdateringen av ekspertsystemer og deres kunnskapsbaser mye enklere. De tungvinte postforsendelsene fra 1980-tallet er en saga blott.

Internett gjør også at ekspertsystemer kan være langt mer tilgjengelige for sine brukere. Nettbaserte ekspertsystemer kan tas i bruk fra over alt hvor Internett er tilgjengelig.

Det ble påpekt at Internett gir noen muligheter for å forbedre kunnskapsakvisisjonen. De mest ambisiøse idéene går ut på å gi ekspertsystemer muligheten til selv å finne relevant informasjon over Internett. Men dette ligger enda i fremtiden. Som en mer håndfast bruk av Internett i kunnskapsakvisisjonen så vi at noen systemutviklere brukte Internett som et kommunikasjonsverktøy ovenfor ekspertene. Det var imidlertid noe uklart hvorvidt dette bidro til å lette denne tungvinte prosessen.

Vi så at det også finnes muligheter for å gjøre ekspertsystemer mer robuste ved hjelp av Internett. En mulighet er å la ekspertsystemer dele hverandres kunnskapsbaser, eller bruke felles kunnskapsbaser. Slik kan systemer supplere manglende regler gjennom Internett. En annen mulig tilnærming til dette problemet er opprettelsen av store kunnskapsbaser med allmennkunnskap som ekspertsystemer så kan få tilgang til. Vi så også en helt konkret løsning på skjørhetsproblemet der man, via Internett, falt tilbake på en menneskelig ekspert dersom systemet ikke kom frem til noen løsning.

Når det gjelder den problematiske usikkerhetshåndteringen løser Internett ikke dette problemet i og for seg. Dette problemet bunner som vi har sett i

egenskaper ved det menneskelige tankesett. Men kanskje kan Internett tenkes å hjelpe ekspertsystemer med å bruke statistiske verktøy for problemer der dette er relevant.

5.2 Konklusjon

Hovedformålet med denne oppgaven har vært å se på hvordan Internett kan bidra til å løse ekspertsystemfeltets klassiske problemer.

Det er åpenbart at Internettets største bidrag er å gjøre ekspertsystemene mye enklere å holde oppdatert enn det som var tilfellet på 1980-tallet. En helt vesentlig forutsetning for å kunne bruke ekspertsystemer er jo at den nyeste og riktigste kunnskapen benyttes av kunnskapsbasene. Den internettbaserte oppdateringen av ekspertsystemers kunnskap er derfor et betydelig fremskritt for ekspertsystemtradisjonen og bidrar til å gjøre denne teknologien langt mer anvendbar.

Det andre store bidraget Internett gir er å øke tilgjengeligheten til ekspertsystemene slik at de kan brukes fra langt flere steder enn tidligere. Denne økte tilgjengeligheten er viktig siden man i ekspertsystemtradisjonen er opptatt av å gjøre spesialisert kunnskap lettere tilgjengelig. Dette blir ikke gjort bare ved å bygge et ekspertsystem. Ekspertsystemet selv må også kunne nås av dem som har behov for dets kunnskap.

De nye mulighetene for oppdatering og tilgjengelighet er det viktigste Internett gir ekspertsystemtradisjonen. Når det gjelder de andre problemene som er blitt nevnt i denne oppgaven gir ikke internettet like åpenbare løsninger. Men det mangler ikke på forslag på hvordan internettet kan tas i bruk for å få bukt med dem eller i det minste forminske dem. Disse ideene er imidlertid bare i liten grad faktisk utprøvd.

På den annen side forblir en del problemer uløste til tross for Internett. Løsningene som Internett kan bidra med her er stort sett på idéplanet og i liten grad praktisk utprøvd. For det første virker det ikke som om Internetts muligheter hittil har gjort det noe lettere å overføre kunnskap fra eksperter til kunnskapsbaser. For det andre er manglende robusthet fortsatt et problem for ekspertsystemene. Det eneste praktiske forsøket på å få til noe i denne sammenhengen er Fish-expert som, ved hjelp av Internett, styrker robustheten i systemet ved å falle tilbake på menneskelig eksperter. For det tredje hjelper ikke Internett ekspertsystemene til å håndtere usikker informasjon på noen bedre måte enn før.

Jeg vil derfor konkludere med at Internett løser to av problemene ekspertsystemene hadde på 1980-tallet, oppdateringsproblemet og tilgjengelighetsproblemet. Disse problemene har imidlertid vært store problemer for ekspertsystemtradisjonen og løsningen av dem gjør at ekspertsystemene opplever en aldri så liten gjenfødelse.

Referanser

Brehmer, B. (1980), 'In one word: Not from experience', *Acta Psychologica* **45**, 223–241.

Buchanan, B. G., Davis, R. & Feigenbaum, E. A. (2006), Expert systems: A perspective from computer science, *in* K. A. Ericsson, N. Charness, R. R. Hoffman & P. J. Feltovich, eds, 'The Cambridge Handbook of Expertise and Expert Performance', Cambridge University Press, pp. 87–103.

Buchanan, B. & Shortliffe, E. (1984), *Rule-Based Expert Systems: The MYCIN experiments of the Stanford Heuristic Programming Project*, Addison-Wesley, Reading, MA.

Camerer, C. F. & Johnson, E. J. (1991), The process-performance paradox in expert judgement. how can experts know so much and predict so badly?, *in* A. Ericsson & J. Smith, eds, 'Toward a general theory of expertise', Cambridge University Press, pp. 195–217.

Carroll, B. (1987), 'Artificial intelligence expert systems for clinical diagnosis: Are they worth the effort?', *Behavioral science* **32**, 274–292.

Davis, R. (1982), 'Expert systems: Where are we? and where do we go from here?', *AI Magazine* **3**(2), 3–22.

Dawes, R. M. & Corrigan, B. (1974), 'Linear models in decision making', *Psychological Bulletin* **81**(2).

Dawes, R. M., Faust, D. & Meehl, P. E. (1989), 'Clinical versus actuarial judgment', *Science* **243**, 1668–1674.

Duan, Y., Edwards, J. S. & Xu, M. X. (2005), 'Web-based expert systems: benefits and challenges', *Inf. Manage.* **42**(6), 799–811.

Duan, Y., Fu, Z. & Li, D. (2003), 'Toward developing and using web-based tele-diagnosis in aquaculture', *Expert Syst. Appl.* **25**(2), 247–254.

Duan, Y., Fu, Z., & Li, D. (2002), 'Fish-expert: a web-based expert system for fish disease diagnosis', *Expert Systems with Applications* **23**, 311–320.

Durkin, J. (1996), 'Expert systems: A view of the field', *IEEE Expert: Intelligent Systems and Their Applications* **11**(2), 56–63.

Ericsson, K. A. & Charness, N. (1994), 'Expert performance. its structure and aquisition', *American Psychologist* **49**(8), 725–747.

Grove, R. (2000), 'Internet-based expert systems', *Expert Systems* **17**(3).

Hammond, K. H. (1996), *Human Judgement and Social Policy*, Oxford University Press.

Hastie, R. & Dawes, R. M. (2001), *Rational Choice in an Uncertain World. The Psychology of Judgement and Decision Making*, Sage Publications Inc.

Heathfield, H. (1999), 'The rise and 'fall' of expert systems in medicine', *Expert Systems* **16**(3).

Huang, M.-J. & Chen, M.-Y. (2007), 'Integrated design of the intelligent web-based chinese medical diagnostic system (cmds) - systematic development for digestive health', *Expert Syst. Appl.* **32**(2), 658–673.

Kengpol, A. & Wangananon, W. (2006), 'The expert system for assessing customer satisfaction on fragrance notes: Using artificial neural networks', *Comput. Ind. Eng.* **51**(4), 567–584.

McCarthy, J. (1984), Some expert systems need common sense, *in* 'Proc. of a symposium on Computer culture: the scientific, intellectual, and social impact of the computer', New York Academy of Sciences, New York, NY, USA, pp. 129–137.

Meehl, P. E. (1954), *Clinical versus Statistical Prediction: A Theoretical Analysis and a Review of the Evidence*, Minneapolis: University of Minnesota Press.

Norvig, P. (1992), *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Schwartz, W. B. & R. S. Patil, P. S. (1987), 'Artificial intelligence in medicine: Where do we stand?', *New England Journal of Medicine* **316**, 685–688.

Simpson, J., Kingston, J. & Molony, N. R. (1999), 'Internet-based decision support for evidence-based medicine', *Knowl.-Based Syst.* **12**(5-6), 247–255.

A Scientia: Et lite ekspertsystemskall i Common Lisp

Som en praktisk programmeringsøvelse og for å få en følelse for hvordan ekspertsystemteknologien fungerer er det også en del av denne oppgaven å lage et lite ekspertsystemskall.

Programmeringsspråket som ble valgt for denne oppgaven er Common Lisp. Common Lisp ble i all hovedsak valgt for å få øvelse med dette språket.

Opprinnelig var planen å lage et nettbasert ekspertsystemskall for så og si å demonstrere poenget med slike nettbaserte systemer. Dette lot seg dessverre ikke gjennomføre siden det ikke ble gitt tillatelse til å kjøre en Lisp server på en av Universitetets maskiner. Jeg hadde heller ikke tilgang til en privat server. Det ble derfor valgt en nødløsning. Isteden for at man kan koble seg opp mot en server og kjøre skallet fra avstand, må brukeren av dette systemet starte en Lisp server (Allegroserve) på sin egen lokale maskin. Lisp serveren kan så kjøre ekspertsystemskallet.

Dette kapittelet har to deler. I den første delen tar vi for oss systemets moduler og beskriver hvordan disse fungerer. I den siste delen av kapittelet ser vi på et kjøringseksempel som viser ekspertsystemskallet i aksjon.

A.1 Systemets moduler

Scientia har to hoveddeler, slutningsmekanismen og utsynet. Disse delene er lagret i hver sin fil. I tillegg finnes filen operasjoner.lisp som inneholder felles funksjoner som brukes av både utsynet og slutningsmekanismen. Her ligger også det lille av kunnskap som systemet er i besittelse av.

Utsynet er ansvarlig for systemets HTML-baserte brukergrensesnitt. Slutningsmekanismen prøver å løse brukerens problemer ved hjelp av den kunnskapen systemet har tilgjengelig. Først skal vi ta undersøke utsynet nærmere.

A.1.1 Utsynet

Utsynet består hovedsakelig av en funksjon, hovedside, som ved hjelp av serveren presenterer ulike brukervalg alt ettersom hvilken tilstand systemet befinner seg i. Dette gjøres gjennom kall til hjelpefunksjoner som tar seg av de enkelte oppgavene.

I utgangspunktet sørger hovedside for at brukeren kan velge hvilke problemer som ønskes løst. Dette skjer gjennom et kall til funksjonen problemvalg. Denne funksjonen lager en rullegardinmeny over mulige valg. Disse valgene er basert på konklusjonene til de reglene som systemet har tilgang til. En konklusjon representerer et problem som systemet kan løse. Har systemet for eksempel en eller flere regler som slutter til konklusjonen diagnose, så er diagnose et mulig valg av problemer.

Når brukeren har oppgitt et problem startes slutningsmekanismen i en egen prosess. Så lenge slutningsmekanismen jobber venter utsynet. Når slutningsmekanismen trenger å stille brukeren et spørsmål legges spørsmålet i en variabel. Utsynet fortsetter så og lager en rullegardinmeny på grunnlag av det slutningsmekanismen vil vite. Det svaret brukeren oppgir blir så ledet tilbake til slutningsmekanismen. Funksjonen hoveddel har ansvaret for dette. Denne

funksjonen har fått navnet sitt fordi den står for hoveddelen av interaksjonen mellom system og bruker.

Gjennom hele denne spørsmålsprosessen er det også utsynets oppgave å sørge for gjennomsiktighet. Dette gjøres ved at brukeren får en begrunnelse for hvorfor det aktuelle spørsmålet blir stilt. Denne begrunnelsen vil være den regelen som systemet forsøker å bekrefte i øyeblikket. I tillegg viser utsynet frem det aktuelle problemet. Dette vil enten være det problemet brukeren selv oppgav, eller et underproblem av dette. Utsynet viser også hvilke regler som er blitt tatt i bruk av slutningsmekanismen så langt i programkjøringen og om de har blitt bekreftet eller forkastet. Funksjonene brukte-regler, begrunnelse og vis-aktuellt-problem er hovedfunksjonene som i hovedsak har ansvaret for systemets gjennomsiktighet.

Når slutningsmekanismen er ferdig viser utsynet, gjennom funksjonen avslutning, enten frem den konklusjonen som slutningsmekanismen kom frem til eller en beskjed om at systemet ikke kom frem til noen konklusjon. Her viser utsynet også hvilke regler som ble brukt i løpet av hele programkjøringen. Herfra kan brukeren også velge å løse et nytt problem eller å avslutte. Å løse et nytt problem innebærer at man blir sendt tilbake til siden der man velger problem. Hvis man velger å avslutte blir man sendt til systemets startside.

A.1.2 Slutningsmekanismen

Slutingsmekanismen fungerer etter baklengs kjeding. Det vil si at den opererer med utgangspunkt i et mål. Den finner alle regler som slutter til dette målet og forsøker å oppnå målet gjennom å bekrefte disse reglene.

Slutningsmekanismen består egentlig bare av en funksjonen trekk-slutning. Denne funksjonen har imidlertid en rekke lokale funksjoner som gjør deler av arbeidet.

Slutningsmekanismen kommer i gang ved at en bruker gjennom brukergrensesnittet oppgir et problem som vedkommende vil løse. Trekk-slutning tar dette problemet og sender det videre til den lokale funksjonen finn-løsning. Problemet som kommer inn vil være et attributt som finnes i systemets kunnskapsbase. Finn-løsning undersøker så dette attributtet. I og med at hele slutningsprosessen er rekursiv må finn-løsning kunne håndtere ulike hendelser avhengig av hvor i rekursjonen man befinner seg. Finn-løsning sjekker derfor først om attributtet allerede finnes lagret i verdibasen. Hvis ikke dette er tilfellet undersøker den om attributtet finnes som konklusjon i en eller flere regler. Hvis ikke spør finn-løsning, gjennom brukergrensesnittet, om verdien til attributtet.

Hvis imidlertid attributtet skulle finnes i verdibasen blir verdien til dette attributtet returnert til funksjonen som kalte finn-løsning. I det tilfellet at attributtet allerede finnes i verdibasen, kom dette kallet fra funksjonen hent-faktisk-verdi. Vi kommer tilbake til denne funksjonen lenger ned.

Hvis det aktuelle attributtet er et slutningsattributt må slutningsprosessen startes ved at man finner alle regler som slutter til dette attributtet. Denne listen over regler sendes så videre til funksjonen regelliste-arbeider. Denne funksjonen må også ta høyde for flere ulike situasjoner som kan oppstå gjennom rekursjonen. Det første regelliste-arbeider gjør er å sjekke om listen over regler er tom. Dette må gjøres siden regelliste-arbeider også kan kalles fra funksjonen premisstester. Ettersom premisstester forkaster regler blir regellisten som oversendes regelliste-arbeider kortere og kortere og til slutt er den tom. Hvis regellisten er tom

undersøker regelliste-arbeider om det aktuelle attributtet systemet arbeider med er det samme som problemet forsøker å løse. Hvis dette er tilfellet betyr det at man ikke har funnet noen konklusjon og regelliste-arbeider returnerer symbolet fiasko. Hvis attributtet ikke er det samme som problemattributtet sørger regelliste-arbeider for å spørre brukeren om attributtets verdi og lagre denne verdien i verdibasen.

Dersom regellisten regelliste-arbeider får oversendt imidlertid ikke er tom fortsetter programkjøringen med et kall på funksjonen regelrytter. Denne funksjonen får oversendt listen over aktuelle regler og forbereder egentlig bare den videre kjøringen ved å oppdatere variabler og datastrukturer som brukes til å gjøre systemet gjennomsiktig for brukeren. Deretter kalles funksjonen premisstester.

Premisstester får oversendt en liste over en gitt regels premisser samt listen over aktuelle regler. Også i denne funksjonen kontrolleres det for forskjellige konklusjoner. Premisstester ser først om premisslisten er tom. Hvis premisslisten er tom betyr det at premisstester i et tidligere kall har jobbet seg gjennom alle premissene og at alle premissene stemte overens med premissene oppgitt i den aktuelle regelen. Det betyr også at regelen slår til. Premisstester returnerer da regelens konklusjon.

Hvis listen over premisser ikke er tom trenger premisstester å finne verdien til det første premisset i listen. Dette overlater premisstester til funksjonen hent-faktisk-verdi som igjen kaller finn-løsning for å få brukerens svar på hvilken verdi attributtet har. Når premisstester får verdien til attributtet sammenligner den verdien brukeren har oppgitt med verdien som kreves for at premisset skal oppfylles. Dersom disse stemmer overens er premisset gyldig og premisstester kaller seg selv med resten av premisslisten som argument. Hvis de to verdiene ikke samsvarer slår ikke premisset til og dermed slår heller ikke den aktuelle regelen til. Denne regelen settes da som ugyldig og premisstester kaller regelliste-arbeider med en regelliste som er en regel kortere.

Slik går hele kjøringen i ring helt til regelliste-arbeider returnerer symbolet fiasko eller helt til det øverste kallet til finn-løsning returnerer en løsningsverdi. I Kroppen til funksjonen trekk-slutning gis denne verdien videre til utsynet gjennom en variabel og slutningsmekanismen er ferdig.

A.2 Et kjøringseksempel

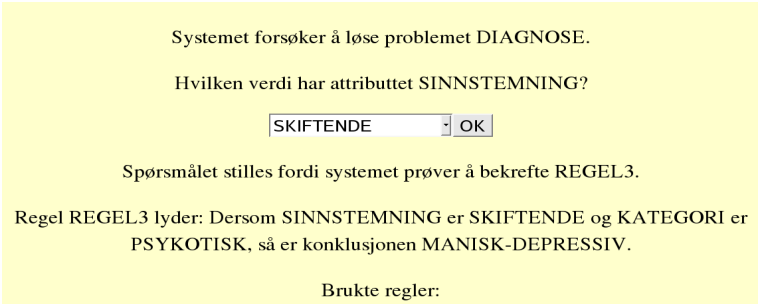
Det første en bruker må gjøre er å velge hvilket problem man vil løse. I dette systemets lille eksempelkunnskapsbase slutter reglene bare til to ulike konklusjoner nemlig diagnose og kategori. Derfor kan man bare velge mellom disse to. Vi ønsker å stille en diagnose, så vi velger problemet diagnose.

Figur 1: Vi velger problemet diagnose.

Slutningsmekanismen settes så i gang på de reglene som konkluderer til

diagnose. Systemet vil forsøke å bekrefte den første og beste regelen.

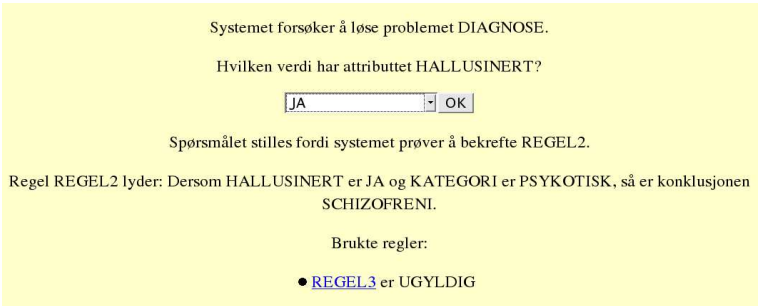
I dette tilfellet er regel 3 den første systemet forsøker å få bekreftet i sitt forsøk på å løse problemet diagnose. Sinnsstemning er det første attributtet i denne regelen, så derfor spør systemet om verdien til sinnsstemning. Systemet begrunner dette spørsmålet med at man forsøker å bekrefte regel 3. Samtidig vises regel 3 slik at man kan se at sinnsstemning er et attributt i denne regelen. Under brukte regler lister systemet opp alle regler som enten er blitt kjent ugyldige eller gyldige. Siden dette er det første spørsmålet er ingen regler blitt verken bekreftet eller avkreftet og derfor står det ingen ting her.



Figur 2: Systemet arbeider med regel 3.

Vi går videre ved å velge sinnsstemningen nedstemt og trykker på knappen merket ok.

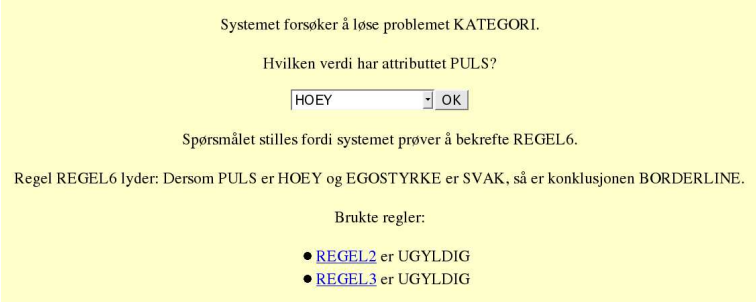
Systemet svarer med å forsøke å bekrefte regel 2 og spør om verdien til attributtet hallusinert. Regel 3 er kjent som ugyldig og står derfor oppført under brukte regler. Vår fiktive pasient har ingen hallusinasjoner så vi velger valget ingen passende valg og går videre.



Figur 3: Regel 2 er i kikkerten. Regel 3 er kjent ugyldig.

Av neste skjermbilde ser vi at regel 2 er funnet ugyldig. Av den første setningen ser vi også at systemet har gått ett nivå lenger ned i kunnskapsbasen og forsøker å løse underproblemet kategori. Systemet forsøker å bekrefte regel 6.

Vår pasient har imidlertid ikke høy puls så vi velger igjen alternativet ingen passende valg. Systemet vil da forsøke å bekrefte regel 5 ved å spørre om verdien til attributtet tilstand. Her oppgir vi verdien kronisk. Dette gjør at regel 5 ikke lenger er aktuell og systemet går videre med regel 4. Regel 4 viser seg å passe på



Figur 4: Systemet forsøker å løse underproblemet kategori. Reglene 2 og 3 er ugyldige

vår pasient og vi kan oppgi verdien surklende til attributtet åndetrett, verdien forvirret til attributtet tale. Siden regel 4 konkluderer med psykotisk er regel 1 oppfylt og systemet kan konkludere med denne regelen som svar på vårt problem:

Systemet lister også i konklusjonen opp alle regler det har vært innom for å finne løsningen på vårt problem. Man kan klikke på hvert regelnavn for å få se den aktuelle regelen. For regel 4 får man for eksempel denne beskrivelsen:

Regel REGEL4 lyder: Dersom TILLSTAND er KRONISK og AANDEDRET er SURKLENDE og TALE er FORVIRRET, så er konklusjonen PSYKOTISK.

B Bruksanvisning for Scientia

For å kunne bruke Scientia må programmet aller først lastes ned. Dette gjøres over nettsiden http://folk.uio.no/steffeh/. Scientia kan enten lastes ned som zip- eller tar fil, alt etter hva man foretrekker.

Pakk så ut filen som ble lastet ned. Resultatet skal bli en mappe som heter scientia. Denne mappen inneholder alt man trenger for å kjøre programmet, naturligvis bortsett fra Allegro Common Lisp (ACL) selv.

Start så en lispinterpret. Programmet krever ACL både på grunn av dennes interne server, Allegroserve, og på grunn av ACLs prosesshåndtering. Scientia vil derfor neppe kunne kjøre under andre lispimplementasjoner. Universitetet i Oslo har imidlertid en lisens til ACL. For øvrig kan en gratis prøveversjon av ACL lastes ned fra Franz inc. sine hjemmesider (http://franz.com/).

Det anbefales ikke å starte programmet under Windows av flere årsaker. For det første bruker Scientia en del eksperne pakker og bruker ASDF systemet for å håndtere disse. ASDF bruker symbolske lenker for å finne hvor disse pakkene ligger. Windows støtter ikke symbolske lenker. Hvis man derfor vil kjøre programmet fra Windows må man skrive inn de absolutte stinavnene i oppstart.lisp slik at stinavnene stemmer overens med det lokale systemet. I tillegg til dette blokkerer Windows innebygde brannmur serveren Allegroserve.

Start så en lispinterpret, for eksempel gjennom Emacs og SLIME. I interpreten last inn filen oppstart.lisp med funksjonen load. Denne filen vil, når den blir evaluert av Allegro Common Lisp, starte opp Allegroserve og kjøre programmet på denne. oppstart.lisp ligger i katalogen scientia. Hvis derfor scientia ligger på rotkatalogen blir load seende slik ut

```
(load ‘‘scientia/oppstart.lisp’’)
```

Dette burde gjøre susen og ekspertsystemskallet kjører nå. Scientia bruker port 2001 i utgangspunktet. Ønsker man å endre på dette kan man gjøre det gjennom å endre følgende passasje i oppstart.lisp:

```
(net.aserve:start :port 2001)
```

For å faktisk bruke skallet må man åpne en nettleser og oppgi adressen

```
http://localhost:2001/skall
```

Da kommer man til systemets startside.

C Kildekode

C.1 operasjoner.lisp

```
1
2  ;Funksjonen i-verdibase? Sjekker om et attributt er lagret i
3  ;verdibasen.
4  ;Returnerer: et attributts verdi.
5  ;Kaller: -
6  ;Kalles av: finn-løsning.
7  (defun i-verdibase? (attributt)
8    (gethash attributt *verdibase*))
9
10 ;Funksjonen liste-over-konklusjoner lager en liste over alle
11 ;konklusjonene som finnes i regelmengden.
12 ;Returnerer: en liste med alle konklusjonene.
13 ;Kaller: seg selv.
14 ;Kalles av: slutningsattributt? og problemvalg (i utsyn).
15 (defun liste-over-konklusjoner (regler liste)
16   (let ((y (if (null regler)
17                liste
18                (let ((x (regel-konklusjon (car regler))))
19          (liste-over-konklusjoner (cdr regler)
20                                   (cons (car x) liste))))))
21     (remove-duplicates y :test #'equal)))
22
23 ;Funksjonen slutningsattributt? Sjekker om det finnes
24 ;regler som slutter til dette attributtet.
25 ;Returner: t hvis regler slutter til attributtet.
26 ;Returnerer nil hvis ikke.
27 ;Kaller: -
28 ;Kalles av: finn-løsning.
29 (defun sluttningsattributt? (attributt regler)
30
31   (defun iterator (liste)
32     (cond
33       ((null liste) nil)
34       ((equal (car liste) attributt) t)
35       (t (iterator (cdr liste)))))
36
37   (iterator (liste-over-konklusjoner regler nil)))
38
39 ;Funksjonen har-attributtnavn-i-konklusjon lager en liste over
40 ;alle regler som slutter til et gitt attributt.
41 ;Returnrer: En liste over regler.
42 ;Kaller: seg selv.
43 ;Kalles av: finn-løsning.
44 (defun har-attributtnavn-i-konklusjon (attributt regler liste)
45   (cond
46     ((null regler) liste)
47     ((equal attributt (car (regel-konklusjon (car regler))))
48      (har-attributtnavn-i-konklusjon attributt (cdr regler)
49                                      (cons (car regler) liste)))
50     (t (har-attributtnavn-i-konklusjon attributt
51                                        (cdr regler) liste))))
52
53 ;Funksjonen finn-verdier finner frem til alle mulige verdier et
54 ;attributt kan ha.
55 ;Returnerer: en liste over mulige verdier.
56 ;Kaller: seg selv.
57 ;Kalles av: still-spoersmaal.
58 (defun finn-verdier (attributt regel-liste verdiliste)
```

```
59    (if (null (car regel-liste))
60        verdiliste
61        (let ((premiss-verdi (finn-verdi attributt
62                                         (regel-premisser
63                                          (car regel-liste)))))
64          (if (null premiss-verdi)
65              (finn-verdier attributt (cdr regel-liste) verdiliste)
66              (finn-verdier attributt (cdr regel-liste)
67                            (cons premiss-verdi verdiliste))))))
68
69  ;Funksjonen finn-verdi henter verdier som tilhører et
70  ;gitt attributt.
71  ;Returnerer: en verdi til et gitt attributt.
72  ;Kaller: seg selv.
73  ;Kalles av: finn-verdier
74  (defun finn-verdi (attributt liste) ;&optional test)
75    (cond
76      ((null liste) nil)
77      ((eql attributt (car (car liste))) (car (cdr (car liste))))
78      (t (finn-verdi attributt (cdr liste)))))
79
80  ;Funksjonen finn-regel finner frem til en regel med et gitt navn.
81  ;Returnerer: den aktuelle regelen hvis den finnes. Nil hvis ikke.
82  ;Kaller: -
83  ;Kalles av: regler (i utsyn), finn-konk-regel (i utsyn).
84  (defun finn-regel (navn)
85    (dolist (regel regler)
86      (if (eql navn (regel-navn regel))
87          (return regel)
88          nil)))
89
90  ;Funksjon nullstill nullstiller alle skallets verdier.
91  ;Returnerer: -
92  ;Kaller: -
93  ;Kalles av: ferdig (i utsyn).
94  (defun nullstill ()
95    (clrhash *verdibase*)
96    (clrhash *historie*)
97    (setf problem nil)
98    (setf spoersmaal nil)
99    (setf svar nil)
100   (setf konk nil)
101   (setf aktuell-regel nil)
102   (setf prosess-startet? nil))
103
104 ;Systemets regler
105 (defstruct regel premisser konklusjon navn)
106
107 (setf regler (list
108 (setf regel1
109       (make-regel :premisser
110                   '((sinnstemning nedstemt) (kategori psykotisk))
111                   :konklusjon '(diagnose psykotisk-depressiv)
112                   :navn 'regel1))
113
114 (setf regel2
115       (make-regel :premisser
116                   '((hallusinert ja) (kategori psykotisk))
117                   :konklusjon '(diagnose schizofreni)
118                   :navn 'regel2))
119
120 (setf regel3
```

```
121         (make-regel :premisser
122                     '((sinnstemning skiftende) (kategori psykotisk))
123                     :konklusjon '(diagnose manisk-depressiv)
124                     :navn 'regel3))
125
126 (setf regel4
127         (make-regel :premisser
128                     '((tillstand kronisk) (aandedret surklende)
129                       (tale forvirret))
130                     :konklusjon '(kategori psykotisk)
131                     :navn 'regel4))
132
133 (setf regel5
134         (make-regel :premisser
135                     '((tillstand akutt) (egostyrke stor) (tale klar))
136                     :konklusjon '(kategori nevrotisk)
137                     :navn 'regel5))
138
139 (setf regel6
140         (make-regel :premisser
141                     '((puls hoey) (egostyrke svak))
142                     :konklusjon '(kategori borderline)
143                     :navn 'regel6))
144 ))
```

C.2 slutningsmekanisme.lisp

```
1
2  ;---------------- Verdibase ----------------
3
4  ;Verdibasen vet hvilke atributter som er blitt gitt
5  ;hvilke verdier.
6  (defparameter *verdibase* (make-hash-table))
7
8  ;------------ Aktuellt problem ------------
9  ;Holder rede på hvilket problem man til enhver tid
10 ;holder på med å løse.
11 (setf aktuellt-problem nil)
12
13 ;---------------- Historie ----------------
14 ;Holder rede på rekkefølgen regler er blitt brukt på.
15 ;Og om disse ble verifisert eller falsifisert.
16 (defparameter *historie* (make-hash-table))
17
18 ;---------------- Aktuell-regel ----------
19 ;Rekkefølgen av reglene som er blitt kallt.
20 ;Legges til ved aktivering og fjernes ved/etter evaluering
21 ;uansett utfall.
22 (setf aktuell-regel nil)
23
24 ;Variabel som kommuniserer spørsmålet slutningsmekanismen
25 ;trenger å stille til utsynet.
26 (setf spoersmaal nil)
27 ;Svaret utsynet får av brukeren.
28 (setf svar nil)
29 ;Dersom man kommer frem til en konklusjon settes
30 ;denne variabelen.
31 (setf konk nil)
32
33 ;Tråden som skal kjøre trekk-slutning.
34 (setf trekk-slutning-traad
35       (mp:make-process :name "Trekk slutning"))
```

```
36
37  ;Portene som styrer samarbeidet mellom utsynet og
38  ;slutningsalgoritmen.
39  (setf hent-faktisk-verdi-port (mp:make-gate nil))
40  (setf still-spoersmaal-port (mp:make-gate nil))
41  (setf utsynport (mp:make-gate nil))
42
43  ;Funksjonen trekk-slutning inneholder hele slutningsalgoritmen
44  ;i form av mange lokale funksjoner.
45  ;Returnerer: Intet, men setter konklusjonsvariabelen som
46  ;utsynet bruker.
47  ;Kaller: finn-løsning (lokal funksjon).
48  ;Kalles av: hoveddel i utsyn.
49  (defun trekk-slutning (probl)
50    (let ((problem probl))
51      (labels
52
53          ;Funksjonen finn-løsning. Denne funksjonen får inn et
54          ;attributt og sjekker om dette allerede ligger i
55          ;verdibasen eller om det er et slutningsattributt. Hvis
56          ;ingen av disse mulighetene er tilfellet spør
57          ;funksjonen om verdien til attributtet.
58          ;Returnerer: -
59          ;Kaller: regelliste-arbeider,
60          ;har-attributtnavn-i-konlusjon, i-verdibase,
61          ;slutningsattributt? og still-spoersmaal.
62          ;Kalles av: trekk-slutning og hent-faktisk-verdi.
63          ((finn-løsning (attributt)
64             (let ((x (i-verdibase? attributt)))
65               (cond
66                 ;Hvis attributtet finnes i verdibasen oppgis verdien
67                 ;som er lagret der.
68                 ((not (null x))
69                  (if (not (mp:gate-open-p hent-faktisk-verdi-port))
70                      (and (setf svar x)
71                           (mp:open-gate hent-faktisk-verdi-port))
72                      x))
73                 ;Er attributtet et slutningsattributt må vi lage en
74                 ;liste over regler med denne
75                 ;konklusjonen og sende listen videre til
76                 ;regelliste-arbeider.
77                 ((sluttningsattributt? attributt regler)
78                  (progn
79                    (setf aktuellt-problem attributt)
80                    (let ((y (regelliste-arbeider
81                              attributt
82                              (har-attributtnavn-i-konklusjon
83                               attributt regler nil))))
84                      (when
85                          (not
86                           (mp:gate-open-p hent-faktisk-verdi-port))
87                        (setf svar y)
88                        (mp:open-gate hent-faktisk-verdi-port)))))
89                 (t (still-spoersmaal attributt))))
90               (format t "Finn-løsning er ferdig.~% Portstatus:~%
91  Hjemmeport:~A~% Borteport:~A~% Utsynport:~A~%"
92                       hent-faktisk-verdi-port
93                       still-spoersmaal-port utsynport))
94
95          ;Funksjonen regelliste-arbeider får en regelliste med de
96          ;reglene som slutter til det aktuelle problemet. Dersom
97          ;denne listen er tom avgjør funksjonen om
```

```
98          ;slutningsalgoritmen avslutter med fiasko eller om man
99          ;stiller spørsmål.
100         ;Returnerer: symbolet fiasko under noen omstendigheter.
101         ;Kaller: still-spoersmaal, regelrytter.
102         ;Kalles av: finn-løsning, premisstester.
103         (regelliste-arbeider (attributt regelliste)
104           (if (null regelliste)
105               (if (eq attributt problem)
106                   'fiasko
107                   (still-spoersmaal attributt))
108               (regelrytter regelliste attributt)))
109
110         ;Funksjonen regelrytter forbereder den videre kjøringen
111         ;ved å
112         ;oppdatere aktuell-regel og historie.
113         ;Returnerer: -
114         ;Kaller: premisstester.
115         ;Kalles av: regelliste-arbeider.
116         (regelrytter (regelliste attributt)
117           (let* ((regel (car regelliste))
118                  (prem (regel-premisser regel)))
119             ;Den regelen vi jobber med nå pushes på stakken:
120             (push regel aktuell-regel)
121             ;Regelen legges inn i historie, uten noen verdi:
122             (setf (gethash (regel-navn regel) *historie*) nil)
123             (premisstester prem regelliste attributt)))
124
125         ;Funksjonen premisstester arbeider med en regels
126         ;premisser.
127         ;Returnerer: konklusjonen til en gyldig regel.
128         ;Kaller: seg selv, regelliste-arbeider,
129         ;hent-faktisk-verdi.
130         ;Kalles av: regelrytter, seg selv.
131         (premisstester (premiss regelliste attributt)
132           (if (null premiss)
133               ;Hvis premisset er oppbrukt vil det si at
134               ;regelen er gyldig.
135               ;Historie og aktuell-regel oppdateres tilsvarende
136               ;og konklusjonen returneres.
137               (let ((foerste-regel (car regelliste)))
138                 (setf
139                  (gethash
140                   (regel-navn foerste-regel) *historie*)
141                  'gyldig)
142                 (pop aktuell-regel)
143                 (car (cdr (regel-konklusjon foerste-regel))))
144               ;Hvis premisset ikke er tomt forsøker vi å
145               ;finne verdien til det aktuelle
146               ;ukjente leddet i premisset.
147               (let* ((navn (car (car premiss)))
148                      (verdi (car (cdr (car premiss))))
149                      (faktisk-verdi (hent-faktisk-verdi navn)))
150                 ;Hvis verdien utenfra stemmer overens med
151                 ;verdien i premisset gjør
152                 ;vi et rekursivt kall med resten av premissene.
153                 (if (equal verdi faktisk-verdi)
154                     (premisstester
155                      (cdr premiss) regelliste attributt)
156                     ;Hvis ingen overenstemmelse kan vi slutte
157                     ;at denne regelen ikke er gyldig.
158                     (progn
159                       (setf
```

```
160                          (gethash (regel-navn
161                                    (car regelliste)) *historie*)
162                          'ugyldig)
163                         (pop aktuell-regel)
164                         (regelliste-arbeider attributt
165                                              (cdr regelliste)))))))
166
167           ;Funksjonen still-spoersmaal setter spoersmaal variabelen
168           ;og vekker den sovende utsyntråden klarsignal til å
169           ;fortsette.
170           ;Returnerer: -
171           ;Kaller: -
172           ;Kalles av: finn-løsning, regelliste-arbeider.
173           (still-spoersmaal (attributt)
174             (setf spoersmaal
175                   (list attributt
176                         (finn-verdier attributt regler nil)))
177             ;La utsynet fortsette. Spoersmaal er satt.
178             (mp:open-gate utsynport)
179             (mp:process-run-function
180              "Venteprosess"
181              #'(lambda ()
182                  (loop
183                   (mp:process-wait "Vent på svar fra utsyn."
184                                    #'mp:gate-open-p
185                                    still-spoersmaal-port)
186                   (setf (gethash attributt *verdibase*) svar)
187                   (mp:close-gate still-spoersmaal-port)
188                   (mp:open-gate hent-faktisk-verdi-port)
189                   (return)))))
190
191           ;Funksjonen hent-faktisk-verdi forsøker å finne ut av
192           ;en verdi ved et kall på finn-løsning.
193           ;Returnerer: svaret man kommer frem til.
194           ;Kaller: finn-løsning.
195           ;Kalles av: premisstester.
196           (hent-faktisk-verdi (navn)
197             (let* ((x (finn-løsning navn))) ;let* nødvendig?
198               (loop
199                ;Hvis vi ikke venter her blir svar en prosess.
200                (mp:process-wait "Venter på finn-løsning."
201                                 #'mp:gate-open-p
202                                 hent-faktisk-verdi-port)
203                (mp:close-gate hent-faktisk-verdi-port)
204                (if (not (null x))
205                    (return x)
206                    (let ((temp svar))
207                ;For å slippe å nullstille svar i en annen funksjon.
208                      (setf svar nil)
209                      (return temp)))))))
210
211         ;Kroppen til funksjonen trekk-slutning.
212         (if (null problem)
213             nil
214             (let ((konklusjon (finn-løsning problem)))
215          ;Hvis konklusjonen er null har vi ikke funnet noen løsning.
216               (if (null konklusjon)
217                   (setf konk svar)
218                   (setf konk konklusjon))
219               (when (mp:gate-open-p hent-faktisk-verdi-port)
220                 (mp:close-gate hent-faktisk-verdi-port)))))))
221 ;-------------- Trekk-slutning slutt --------------
```

C.3 utsyn.lisp

```
1
2
3  ;Funksjonen hovedside har som oppgave å
4  ;oppdatere hovedsiden i html
5  ;brukergrensesnittet.
6  ;Kaller: avslutning, problemvalg, hoveddel.
7  ;Kalles av: startsiden (start.html).
8  (defun hovedside (request entity)
9    (with-http-response
10       (request entity :content-type "text/html"
11                :response *response-found*)
12     (with-http-body (request entity)
13       (with-html-output ((request-reply-stream request))
14         (cond
15
16           ;Her er vi ferdige, enten det er funnet
17           ;en løsning eller ikke:
18           ((not (null konk))
19            (avslutning))
20
21           ((null (request-query request))
22            (problemvalg))
23
24           ((not (null (request-query-value "sidevalg" request)))
25            (hoveddel request))
26
27           ((not (null (request-query-value "verdi" request)))
28            (progn ;Verdien settes inn i variabelen svar:
29              (setf svar
30                    (intern
31                     (request-query-value "verdi" request) :web))
32              (mp:open-gate still-spoersmaal-port)
33
34              (html
35                (:mal
36                 (:title "Vennligst vent")
37                 (:meta
38                  :http-equiv "REFRESH"
39                  :content
40                  "0; url=/hovedside?sidevalg=verdi-spoersmal")))
41              )))))))
42
43 ;Funksjonen ferdig avslutter ekpspertsystemsesjonen ved å
44 ;kalle nullstill og ved å sende brukeren enten ut av systemet
45 ;eller til siden der man kan velge problem.
46 ;Kaller: nullstill.
47 ;Kalles av: avslutning (via html form).
48 (defun ferdig (request entity)
49   (with-http-response
50       (request entity
51                :content-type "text/html"
52                :response *response-found*)
53     (with-http-body (request entity)
54       (with-html-output ((request-reply-stream request))
55         (nullstill)
56         (html
57           (:mal
58            (:title "Vennligst vent")
59            (princ
60             (if (not
```

```
61                    (null
62                     (request-query-value "avslutt" request)))
63                   (html
64                     (:meta :http-equiv "REFRESH"
65                            :content "0; url=/skall"))
66                   (html
67                     (:meta :http-equiv "REFRESH"
68                            :content "0; url=/hovedside"))))))))))
69
70  ;Funksjonen avslutning viser frem systemets konklusjon og gir
71  ;mulighet for å løse et nytt problem eller for å avslutte.
72  ;Kaller: brukte-regler.
73  ;Kalles av: hovedside.
74  (defun avslutning ()
75    (html
76      (:mal
77       (:title "Konklusjon")
78       (princ
79        (if (eq konk 'fiasko)
80            (html
81              (:p "Systemet har ikke klart å finne en
82  løsning på ditt problem."))
83            (html
84              (:p "Konklusjonen er " konk)
85              (begrunnelse (finn-konk-regel konk) t))))
86       (brukte-regler)
87       (:form :method "POST"
88              :action "/ferdig"
89              (:input :type "submit"
90                      :value "Løs et nytt problem"
91                      :name "ny-start"))
92       (:form :method "POST"
93              :action "/ferdig?avslutt=true"
94              (:input
95               :type "submit"
96               :value "Avslutt"
97               :name "avslutt")))))
98
99  ;Funksjonen problemvalg er ansvarlig for menyen
100 ;der brukeren kan velge hvilke problemer som skal løses.
101 ;Kaller: rullegardin.
102 ;Kalles av: hovedside.
103 (defun problemvalg ()
104   (let ((konklusjoner (liste-over-konklusjoner regler nil)))
105             (html
106               (:mal
107                (:title "Hvilke problemer vil du løse?")
108                (princ
109                 (if (not (null konklusjoner))
110                     (html
111                       (:p "Hvilke problemer vil du løse?")
112                       (rullegardin konklusjoner
113                                    "/hovedside"
114                                    "problemer"
115                                    t
116                                    "problem-spoersmaal"
117                                    "sidevalg"
118                                    :ingen-p-valg
119                                    nil))
120                   (html
121                     (:p "Det finnes ingen
122 kunnskapsbase i systemet!")))))))))
```

```
123
124 ;Funksjonen hoveddel står for de html elementene som lar
125 ;brukeren besvare spørsmål fra systemet.
126 ;Kaller: rullegardin og brukte-regler.
127 ;Kalles av: hovedside.
128 (defun hoveddel (request)
129   (when
130       (equal
131        (request-query-value "sidevalg" request)
132        "problem-spoersmaal")
133     ;Trekk-slutning startes i en egen tråd og
134     ;får med seg problemet som parameter.
135     (mp:process-preset trekk-slutning-traad
136                        #'trekk-slutning
137                        (intern
138                         (request-query-value "problemer" request)
139                         :web))
140     (mp:process-enable trekk-slutning-traad))
141
142   ;Så må utsynet vente på slutningsmekanismen.
143   (loop
144    (mp:process-wait "Venter på trekk-slutning"
145                     #'mp:gate-open-p
146                     utsynport)
147    (return))
148   (mp:close-gate utsynport)
149
150   ;Her må man finne ut hvilket attributt man vil spørre om.
151   (let* ((attributt (car spoersmaal))
152          (verdier (car (cdr spoersmaal))))
153     (html
154       (:mal
155        (:title "Hvilken verdi har attributtet?")
156        (html
157          (vis-aktuellt-problem)
158          (:p "Hvilken verdi har attributtet " attributt "?")
159          (rullegardin verdier "/hovedside" "verdi")))))
160   (begrunnelse (car aktuell-regel) nil t)
161   (brukte-regler)
162   (setf spoersmaal nil))
163
164 ;Funksonen regler lager en html-side som beskriver en regel.
165 ;Kaller: begrunnelse og finn-regel.
166 ;Kalles av: listemaker (via html-lenke).
167 (defun regler (request entity)
168   (with-http-response (request entity)
169     (with-http-body (request entity)
170       (with-html-output ((request-reply-stream request))
171         (let ((regelnavn (request-query-value "regel" request)))
172           (html
173             (:mal
174              (:title (:format "Regelen ~A" regelnavn))
175               (:p (princ
176                     (begrunnelse
177                      (finn-regel
178                       (intern regelnavn :web))))))))))))
179
180 ;Funksjonen vis-aktuellt-problem gjør akkurat det navnet sier.
181 ;Kaller: -
182 ;Kalles av: hoveddel.
183 (defun vis-aktuellt-problem ()
184   (html
```

```
185       (:p
186        "Systemet␣forsøker␣å␣løse␣problemet␣"
187        aktuellt-problem "."))))
188
189   ;Funksjonen begrunnelse begrunner hvorfor det spørres om et
190   ;attributt og hvis man det finnes en konklusjon,
191   ;hvilken regel som førte til denne.
192   ;Kaller: -
193   ;Kalles av: avslutning og hoveddel.
194   (defun begrunnelse (regel &optional konklusjonen spoersmaal)
195     (let ((navn (regel-navn regel))
196           (premisser (premissutskriver (regel-premisser regel)))
197           (konklusjon (cadr (regel-konklusjon regel))))
198       (html
199         (princ
200          (when konklusjonen
201            (html
202              (:p
203               "Regelen␣som␣sluttet␣til␣denne␣konklusjonen␣var␣"
204               navn "."))))
205          (when spoersmaal
206            (html
207              (:p
208               "Spørsmålet␣stilles␣fordi␣systemet␣prøver␣å␣bekrefte␣"
209               navn "."))))
210          (html
211            (:p "Regel␣" navn "␣lyder:␣Dersom␣" premisser
212                ",␣så␣er␣konklusjonen␣" konklusjon "."))))))
213
214   ;Funksjonen premissutskriver som lager en
215   ;fin streng over en regels premisser.
216   ;Returnerer: en streng med premisser.
217   ;Kaller: seg selv.
218   ;Kalles av: begrunnelse.
219   (defun premissutskriver (premissliste)
220     (when (not (null premissliste))
221       (let* ((prem (car (car premissliste)))
222              (verdi (car (cdr (car premissliste))))
223              (s (format nil "~A␣er␣~A" prem verdi)))
224         (if (null (cdr premissliste))
225             (concatenate 'string s
226                          (premissutskriver
227                           (cdr premissliste)))
228             (concatenate 'string s "␣og␣"
229                          (premissutskriver
230                           (cdr premissliste)))))))
231
232   ;Funksjonen brukte-regler bruker historie-utskrift til
233   ;å lage en liste over hvilke regler som er blitt evaluert
234   ;som gyldig eller ugyldig.
235   ;Kaller: listemaker.
236   ;Kalles av: avslutning og hoveddel.
237   (defun brukte-regler ()
238   ;Vis hvilke regler som ble brukt og om gyldig/ugyldig.
239     (let ((hist (historie-utskrift)))
240       (html
241         (:p "Brukte␣regler:␣")
242         (:ul
243          (mapcar #'listemaker hist)))))
244
245   ;Funksjonen listemaker Lager et element av en html liste
246   ;hvorav det første innholdet av listeelementet er en lenke.
```

```
247 ;Kaller: -
248 ;Kalles av: brukte-regler.
249 (defun listemaker (par)
250   (let* ((regelnavn (car par))
251          (adresse
252           (concatenate
253            'string "http://localhost:2001/regel?regel="
254            (format nil "~A" regelnavn))))
255     (html
256       (:li (:a :href adresse
257                :target "_blank"
258                (:format "~A" regelnavn))
259            (:format "␣er␣~A" (cadr par))))))
260
261 ;Funksjonen historie-utskrift finner frem alle regler merket
262 ;med ugyldig eller gyldig og legger disse i en liste.
263 ;Returnerer: Ovenfor nevnte liste.
264 ;Kaller: -
265 ;Kalles av: brukte-regler.
266 (defun historie-utskrift ()
267   (let ((x))
268     (maphash #'(lambda (k v)
269                  (when (not (eql v 'nil))
270                    (setf x (cons (list k v) x))))
271              *historie*)
272     x))
273
274 ;Funksjonen finn-konk-regel finner frem til regelen
275 ;som konkluderer til oppnådd konklusjon
276 ;Returnerer: regelen som konklusderer til konklusjonen.
277 ;Kaller: -
278 ;Kalles av: begrunnelse.
279 (defun finn-konk-regel (konk)
280   (let ((x))
281     (maphash #'(lambda (k v)
282                  (when (eq v 'gyldig)
283                    (let ((regel (finn-regel k)))
284                      (when
285                          (eq
286                           (cadr
287                            (regel-konklusjon regel))
288                           konk)
289                          (setf x regel)))))
290              *historie*)
291     x))
292
293 ;Funksjonen rullegardin lager html-rullegardinmenyer.
294 ;Kaller: -
295 ;Kalles av: hoveddel og problemvalg.
296 (defun rullegardin (liste aksjon name &optional
297                     hidden hidden-verdi hidden-navn
298                     &key (ingen-p-valg t))
299   (html
300     (:form :method "POST" :action aksjon
301            (:select :name name
302              (princ
303               (loop for x in liste
304                     do (html (:option (:value x)))
305                     finally
306                     (princ
307                      (when ingen-p-valg
308                        (html
```

```
309                             (:option
310                              (:value "Ingen passende valg")))))
311                         (princ
312                          (when (not (null hidden))
313                            (html
314                              (:input :type "hidden"
315                                      :value hidden-verdi
316                                      :name hidden-navn))))
317                         (html
318                           (:input :type "submit"
319                                   :value "OK"
320                                   :name "submit-problem"))))))))
321
322 ;Mal for html-sidene.
323 (define-html-macro :mal ((&key title) &body body)
324   `(:html
325     (:head (:link :rel "stylesheet"
326                   :href "stil.css"
327                   :type "text/css") (:title ,title))
328     (:body
329      ,@body)))
```